

SX-Key Development System Manual
Version 1.0

Warranty

Parallax warrants its products against defects in materials and workmanship for a period of 90 days. If you discover a defect, Parallax will, at its option, repair, replace, or refund the purchase price. Simply call our sales department for an RMA number, write it on the label and return the product with a description of the problem. We will return your product, or its replacement, using the same shipping method used to ship the product to Parallax (for instance, if you ship your product via overnight express, we will do the same).

This warranty does not apply if the product has been modified or damaged by accident, abuse, or misuse.

14-Day Money-Back Guarantee

If, within 14 days of having received your product, you find that it does not suit your needs, you may return it for a refund. Parallax will refund the purchase price of the product, excluding shipping/handling costs. This does not apply if the product has been altered or damaged.

Copyrights and Trademarks

Copyright © 1998 by Parallax, Inc. All rights reserved. Parallax and the Parallax logo are registered trademarks. SX-Key is a registered trademark of Parallax, Inc. Scenix is a registered trademark of Scenix, Inc. PIC is a registered trademark of Microchip Technology, Inc. Windows is a registered trademark of Microsoft Corporation. Other brand and product names are trademarks or registered trademarks of their respective holders.

Disclaimer of Liability

Parallax, Inc. is not responsible for special, incidental, or consequential damages resulting from any breach of warranty, or under any legal theory, including lost profits, downtime, goodwill, damage to or replacement of equipment or property, and any costs of recovering, reprogramming, or reproducing any data stored in or used with Parallax products.

Internet Access

We maintain Internet systems for your convenience. These may be used to obtain software, communicate with members of Parallax, and communicate with other customers. Access information is shown below:

Web:	http://www.parallaxinc.com
Ftp:	ftp.parallaxinc.com
General e-mail:	info@parallaxinc.com
Tech. e-mail:	sxtech@parallaxinc.com

Internet SX Discussion List

We maintain an SX discussion list for people interested in SXs. Many people subscribe to the list, and all questions and answers to the list are distributed to all subscribers. It's a fun, fast, and free way to discuss SX issues. To subscribe to the SX list, send email to majordomo@parallaxinc.com and write *SUBSCRIBE SX* in the body of the message.

This manual is valid with the following software and firmware versions:

Assembler:
SXKey.exe software version 1.01

Firmware:
SX-Key rev. E

The information herein will usually apply to newer versions but may not apply to older versions. New software can be obtained free on our ftp and web site (<ftp.parallaxinc.com>, www.parallaxinc.com). If you have any questions about what you need to upgrade your product, please contact Parallax.

Contents

PREFIX	5
PACKING LIST	5
Skeleton Key	5
Master Key 18	6
Master Key 28	6
INTRODUCTION TO THE SX-KEY HARDWARE	7
QUICK START GUIDE	9
QUICK START INTRODUCTION	9
CONNECTING AND DOWNLOADING (SX-KEY DEMO BOARD).....	9
Configuring HyperTerminal (SX-Key Demo Board).....	11
COMMUNICATING WITH THE VIRTUAL PERIPHERALS IN THE SX CHIP	11
CONNECTING AND DOWNLOADING (SX-KEY QUICKPROTO BOARD).....	12
INSTALLATION OF THE SX-KEY INTERFACE	15
WHAT'S NEW.....	17
FOR THOSE FAMILIAR WITH PIC® MICROCONTROLLERS:	17
Similarities:	17
Differences:	17
FOR THOSE FAMILIAR WITH THE SUPER PARALLAX ASSEMBLER:.....	19
Similarities:	19
Differences:	19
INTRODUCTION TO THE SX-KEY INTERFACE	21
THE MENUS.....	22
The <u>F</u> ile menu:	22
The <u>E</u> dit menu:	23
The <u>R</u> un menu:.....	23
The <u>H</u> elp menu:.....	24
THE WINDOWS.....	24
Print Window	24
Find/Replace Window	25
Debug Windows	25
Debugging:	30
Device Window:.....	31
Configure Window:	32
SX-KEY ASSEMBLER	35

Contents

COMMENTS.....	36
ASSEMBLER DIRECTIVES	36
SYMBOLS.....	50
LABELS.....	51
EXPRESSIONS.....	52
DATA TYPES.....	54
SX SPECIAL FEATURES AND CODING TIPS	55
INTRODUCTION.....	55
PORT CONFIGURATION AND USAGE	55
Port Direction.....	56
Pull-up Resistors	57
Logic Level	58
Schmitt-Trigger	59
Edge Detection	60
Wake-Up (Interrupt) on Edge Detect	62
Comparator.....	64
ALL ABOUT INTERRUPTS.....	66
RTCC Rollover Interrupts	68
CREATING TABLES	71
8-bit Data Tables	71
8-bit or 12-bit Data Tables	72
DEALING WITH CODE PAGES.....	73
Branching Across Pages	73
Calling Across Pages with Jump Tables	74
APPENDIX A: SX FEATURES.....	77
INTRODUCTION.....	77
CPU FEATURES.....	77
PERIPHERAL AND I/O FEATURES	77
APPENDIX B: SX INSTRUCTION SET	79
PARALLAX INSTRUCTION SET SUMMARY	79
SINGLE-WORD INSTRUCTIONS	80
MULTI-WORD INSTRUCTIONS	81
INSTRUCTION SET QUICK REFERENCE.....	82
APPENDIX C: SX-KEY RESERVED WORDS	107
APPENDIX D: SX-KEY ERROR MESSAGES	109
SX-KEY ERROR MESSAGES	109

Contents

APPENDIX E: SX-KEY DEMO/QUICKPROTO BOARD 115

SX-KEY DEMO BOARD FEATURES.....	115
Features	116
VIRTUAL PERIPHERAL SOURCE CODE EXAMPLE.....	116
CONNECTING AND DOWNLOADING.....	116
COMMUNICATION	117
CONFIGURING THE VIRTUAL PERIPHERALS	117
SX-KEY DEMO BOARD SCHEMATIC	118
SX-KEY QUICKPROTO BOARD FEATURES	119
Features	120
CONNECTING AND DOWNLOADING.....	120
SX-KEY QUICKPROTO BOARD SCHEMATIC.....	121

APPENDIX F: SX DATA SHEET 123

PINOUT INFORMATION AND DESCRIPTIONS.....	123
REGISTER MAP STRUCTURE	124
ARCHITECTURE.....	124
INSTRUCTION PIPELINE.....	125
Read-Modify-Write Considerations.....	126
SPECIAL FUNCTION REGISTERS	126
IND - The Indirect Register (\$00)	127
Direct and Indirect Addressing Examples.....	127
RTCC Real Time Clock/Counter and WREG (\$01).....	127
PC - Program Counter (\$02)	127
The Program Counter	128
Jump Instruction	128
Jumping Across Pages.....	128
Call Instruction.....	129
Calling Across Pages	129
Returning from a subroutine	130
The Stack	130
The Push	130
The Pop	131
Stack Overflow	131
Stack Underflow	132
Returns.....	132
STATUS Register (\$03)	132
FSR - File Select Register (\$04)	134
Direct Addressing	135
Indirect Addressing.....	136
The Bank Instruction	136

Contents

PORT CONFIGURATION REGISTERS	137
Port A Registers	137
Port B Registers	138
Port C Registers	141
CONTROL REGISTERS	143
Mode	143
Option	145
Fuses	145
INTERRUPTS	146
Description	146
The Specifics	147
RTCC Interrupt	147
RB0-RB7 Interrupt	147
PERIPHERALS	148
Oscillator Driver	148
XT, LP, and HS Mode	148
External RC Mode	150
Internal RC Mode	151
INDEX	153

Thank you for purchasing the Parallax SX-Key[®] development system. We have done our best to produce a full-featured, yet easy to use development system for the SX microcontrollers. The result is the SX-Key; a very tiny, full-featured programmer and emulator with a Windows[®] 95/98/NT4 interface. We hope you will find this system as enjoyable to use as we do.

Please consult the documentation section of the Parallax web site for the latest SX-Key manual:
<http://www.parallaxinc.com>.

Additionally, download the latest errata sheet from
<http://www.parallaxinc.com/sx/sxerrata.htm>.

Packing List

There are three SX-Key development packages available: the Skeleton Key, the Master Key 18 and the Master Key 28. The packing lists for each of these is listed below:

Skeleton Key

- (1) SX-Key Programmer / Emulator Device
- (1) SX-Key Development System diskette
- (1) SX-Key manual (this manual)
- (1) SX Datasheet

Prefix

Master Key 18

- (1) SX-Key Programmer / Emulator Device
- (1) SX-Key Development System diskette
- (1) SX-Key manual (this manual)
- (1) SX Datasheet
- (1) SX-Key QuickProto Board
- (1) QuickProto Board Power Supply
- (1) 9-pin serial cable
- (2) 18-pin SX chips

Master Key 28

- (1) SX-Key Programmer / Emulator Device
- (1) SX-Key Development System diskette
- (1) SX-Key manual (this manual)
- (1) SX Datasheet
- (1) SX-Key Demo Board
- (1) Demo Board Power Supply
- (1) 9-pin serial cable
- (2) 28-pin SX chips

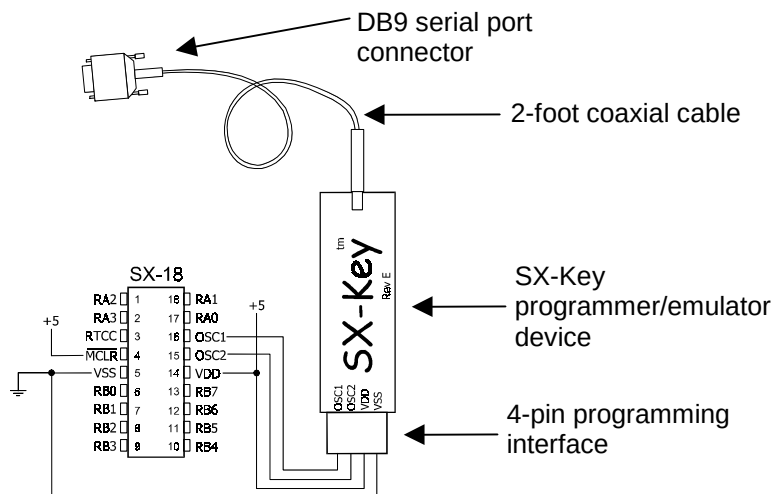
If any items are missing, please let us know.

1: Introduction to the SX-Key Hardware

The SX-Key hardware consists of the programmer/emulator unit, a four-pin programming interface and a two-foot coaxial cable that ends in a standard, female serial port connector (DB9). The serial port connector should be plugged into an available serial port on an IBM-compatible PC. A standard, straight-through serial cable can be used to extend the length of the SX-Key's reach. The four-pin connector on the SX-Key board should be connected to four pins (VSS, VDD, OSC2 and OSC1) of the SX chip itself (see Figure 1.1).

The SX-Key is powered by the target circuit's power supply and programming and debugging takes place over the oscillator pins. If an external crystal, resonator or RC circuit is used, the SX-Key can usually remain connected to the SX chip for programming purposes, without affecting the operation of the circuit. When debugging, the SX chip must not have an external clock source since the SX-Key's internal programmable oscillator must be used. Each SX microcontroller contains the necessary emulator hooks required to perform SX in-circuit emulation. No other supporting chips are necessary for the emulation process. During emulation, the SX-Key provides the oscillator signal to drive the SX microcontroller until such time that a breakpoint is hit or a single step or stop mode is initiated. NOTE: Figure 1.1 shows all the connections necessary to program, debug and run the SX microcontroller.

Figure 1.1: SX-Key Hardware Connection.



Introduction to the SX-Key Hardware

2: Quick Start Guide

Quick Start Introduction

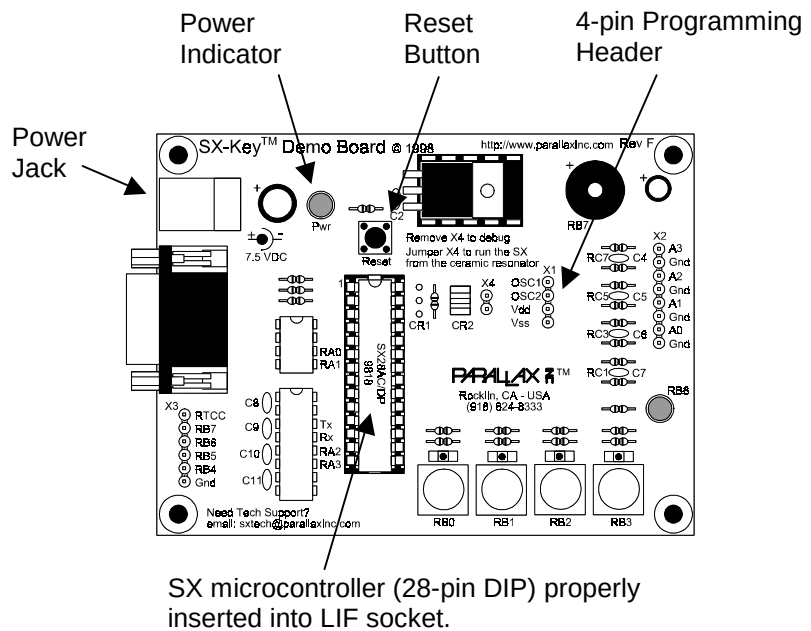
This chapter is a quick start guide to connecting the SX-Key and programming the SX microcontroller. Without even knowing how the SX-Key and the SX chip function, you should be able to obtain satisfactory results from the steps that follow.

This guide is split into two sections, one covers the SX-Key Demo Board and the other covers the SX-Key QuickProto board. Please proceed to the section concerning the board you received (if any).

Connecting and Downloading (SX-Key Demo Board)

In order to get familiar with how the SX-Key Development System works, we'll use the SX-Key Demo Board to program and run a 28-pin SX chip. **This quick start guide section will not work with the SX-Key QuickProto board. Please see the section titled, "Connection and Downloading (SX-Key QuickProto Board)" for more information.**

Figure 2.1: SX-Key Demo Board with SX chip inserted.



Quick Start Guide

Keep in mind that the SX-Key Demo Board is not a programmer; rather the SX-Key is the programmer/emulator device while the demo board is a type of prototyping board. Follow these steps to connect and download a program:

- 1) Plug an SX28AC/DP into the 28-pin LIF socket on the demo board as shown in Figure 2.1. Make sure it is oriented so that the half-moon notch in the chip faces the “Reset” button.
- 2) Connect the SX-Key to a serial cable, and the serial cable to an available serial port (Com: port) on the PC.
- 3) Connect the SX-Key to the 4-pin programming header (labeled X1) with the VSS, VDD, OSC2 and OSC1 indicators lining up with the same indicators on the board. Remove the X4 strap if it is installed.
- 4) Plug the power supply into the demo board and into an available wall outlet. (The power indicator should light up).
- 5) Insert the SX-Key Development System diskette into a floppy drive and use Windows Explorer (or the Run item on the Start menu) to locate and run the sxkey.exe program from the diskette. (The SX-Key window should appear).
- 6) In the SX-Key window, pull down the File menu and select Open (or press CTRL-O). In the browser window that appears, select and open the sxdemo.src file. (The sxdemo.src source code should appear in the SX-Key code window).
- 7) If the serial port (Com: port) you are using is not Com 1, pull down the Run menu, select Configure... and select the proper port from the Configuration window.
- 8) Pull down the Run menu and select Run (or press CTRL-R). (The SX-Key software should assemble the code and begin the programming process).

2: Quick Start Guide

Congratulations! You have just programmed the SX microcontroller with the SX-Key Development System. The SX Virtual Peripheral demonstration program should start running. The piezo speaker will make a clicking noise and the yellow LED should flash on and off. To communicate with and configure the SX Virtual Peripherals, continue with the steps below.

Configuring HyperTerminal (SX-Key Demo Board)

- 9) Disconnect the serial cable from the SX-Key and connect the serial cable to the demo board directly.
- 10) Open the HyperTerminal program. If you are unfamiliar with this program, click on the Start button, select the Run item and type: hypertrm in the Open: field. Note: there is only one “e” in the actual program name. (The HyperTerminal window should appear).
- 11) On the Connection Description window, type: ComPort in the Name: field and click on the OK button. (The Phone Number window should appear).
- 12) In the “Connect Using:” field, select one of the “Direct to Com...” items. **Make sure to select the same Com port number that you used to program the SX in steps 1 through 8.** Click OK when done. (The COM Properties window should appear).
- 13) Select 19200 bits per second, 8 data bits, no parity, 1 stop bit and no flow control. Click the OK button when done.

Communicating with the Virtual Peripherals in the SX chip (SX-Key Demo Board)

Press and release the Reset button on the SX-Key Demo Board. You should hear a clicking noise, the yellow LED should flash and “SX Virtual Peripheral Demo” should appear on your screen. The SX chip is now communicating with HyperTerminal at 19.2K baud over the serial cable.

Your cursor should appear next to a “>” prompt. Now you can type in simple commands to change the behavior of the virtual peripherals.

Quick Start Guide

For example, type: “T6” and press the Enter key. The yellow LED should start blinking faster. Type: “F9” and press the Enter key to increase the frequency on the piezo speaker. (Type: “F0” to stop the speaker). For more information on the SX-Key Demo Board or for a listing of available Virtual Peripheral demo commands, see Appendix E.

Connecting and Downloading (SX-Key QuickProto Board)

In order to get familiar with how the SX-Key Development System works, we'll use the SX-Key QuickProto Board to program and run an 18-pin SX chip. **This quick start guide section will not work with the SX-Key Demo board. Please see the section titled, “Connection and Downloading (SX-Key Demo Board)” for more information.**

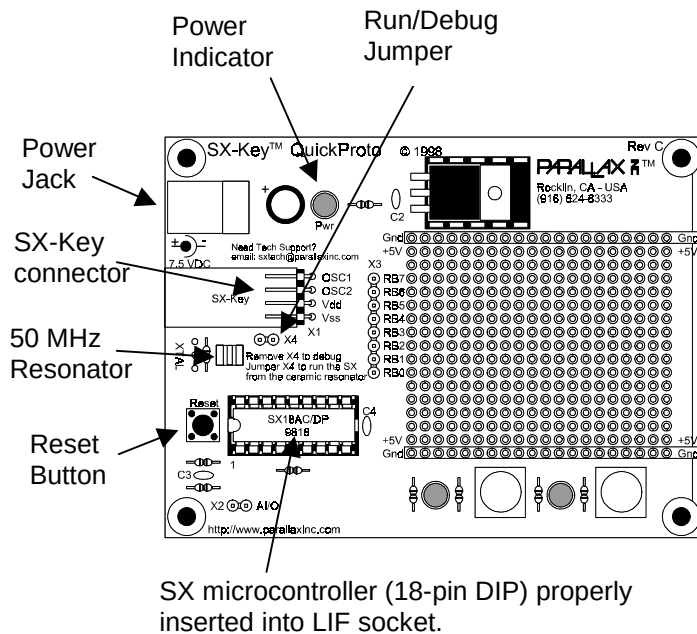


Figure 2.2: SX-Key QuickProto Board with SX chip inserted.

Keep in mind that the SX-Key QuickProto Board is not a programmer; rather the SX-Key is the programmer/emulator device while the QuickProto board is a type of prototyping board. Follow these steps to connect and download a program:

2: Quick Start Guide

- 1) Plug an SX18AC/DP into the 18-pin LIF socket on the QuickProto board as shown in Figure 2.2. Make sure it is oriented so that the half-moon notch in the chip faces the “Reset” button.
- 2) Connect the SX-Key to a serial cable, and the serial cable to an available serial port (Com: port) on the PC.
- 3) Connect the SX-Key to the 4-pin programming header (labeled X1) with the VSS, VDD, OSC2 and OSC1 indicators lining up with the same indicators on the board. Remove the X4 strap if it is installed.
- 4) Plug the power supply into the QuickProto board and into an available wall outlet. (The power indicator should light up).
- 5) Insert the SX-Key Development System diskette into a floppy drive and use Windows Explorer (or the Run item on the Start menu) to locate and run the sxkey.exe program from the diskette. (The SX-Key window should appear).
- 6) In the SX-Key window, pull down the File menu and select Open (or press CTRL-O). In the browser window that appears, select and open the led18.src file. (The led18.src source code should appear in the SX-Key code window).
- 7) If the serial port (Com: port) you are using is not Com 1, pull down the Run menu, select Configure... and select the proper port from the Configuration window.
- 8) Pull down the Run menu and select Run (or press CTRL-R). (The SX-Key software should assemble the code and begin the programming process).

Congratulations! You have just programmed the SX microcontroller with the SX-Key Development System. The program in the SX microcontroller should start running. One of the yellow LEDs should flash on and off.

Quick Start Guide

3: Installation of the SX-Key Interface

The SX-Key Interface (sxkey.exe) consists of the integrated editor, programmer, and debugger software. The following system requirements are a minimum for using the SX-Key Interface:

- 80486 (or higher) IBM or compatible PC;
- Windows 95/98/NT4 operating system;
- 16 Mb of RAM;
- 1 Mb of available hard drive space;
- 3 ½ inch floppy drive;
- 1 available serial port.

(Note: though it is suggested that the SX-Key interface be installed on your hard drive, it is not required. The sxkey.exe program may be run right off the SX-Key Development System diskette).

To install the SX-Key Interface:

1. Insert the SX-Key Development System diskette in an available floppy drive.
2. Run the Windows Explorer program.
3. Create a directory on the hard drive in which you would like to install the software.
4. Copy all files from the floppy drive to the newly created directory.
5. Close Windows Explorer.

To use the SX-Key Interface:

- 1) Run the sxkey.exe program from the directory it was installed into in the procedure above.

Installation of the SX-Key Interface

4: What's New

For those familiar with PIC[®] microcontrollers:

The SX microcontroller has many similarities with Microchip's PIC16C5x series of microcontrollers. The similarities and differences are highlighted here. This is not a complete list. Please refer to the SX Datasheet for more information.

Similarities:

- **Pinout:** The SX-18 and SX-28 microcontrollers have the same pin configuration as the corresponding PIC16C5x devices; although physical package size might differ slightly.
- **RAM Registers:** The RAM registers and RAM banks are identical.
- **Program Space:** The same amount of program space is available (512 x 12, 1024 x 12 and 2048 x 12). This may be expanded upon in future SX microcontrollers.
- **Opcodes:** The same opcodes are supported allowing object code developed for PIC16C5x devices to be programmed into the SX with no changes.
- **Instruction Execution:** The same instruction execution speed is available, in compatibility mode, at up to 20 MHz.

Differences:

- **Programming Pins:** The SX may be programmed using the OSC1 and OSC2 pins. This is the method used by the SX-Key Development System.
- **Program Space:** The program memory is E²Flash capable of being reprogrammed up to 10,000 times.
- **Emulation:** Each SX chip contains the necessary emulation hooks to allow in-circuit emulation. This is achieved with the SX-Key over the oscillator pins. No bondout chips are required and no I/O pin conflicts occur during emulation.
- **Opcodes:** Ten new opcodes are included to allow easy bank and page switching, interrupts with context save/restore, and more.

What's New

- **Instruction Execution:** By selecting the Turbo fuse, the SX chip will execute one instruction per clock, except branches which take three. This option gives you 4 times the speed of Compatibility mode. Additionally, the oscillator of the SX can be driven at up to 50 Mhz. Combined with the Turbo fuse, execution speeds of 50 MIPS can be realized. Faster versions of the SX chip are planned by Scenix.
- **Fuses:** New fuses have been added for functions such as: turbo mode, stack extend, option extend, add/sub with carry and input syncing.
- **Stack:** The SX includes a stack extend fuse to expand the stack from the PIC-compatible 2-levels to 8-levels deep.
- **Interrupts:** Interrupt capability is included with automatic context save/restore to and from built-in shadow registers.
- **Oscillators:** An internal RC oscillator is included that can run at speeds of 31 Khz to 4 Mhz. External oscillators can drive the chip at up to 50 Mhz. Note: SX chips v2.4 (and above) require a resistor across OSC1 and OSC2. See Appendix F for more information.
- **I/O Pins:** All I/O pins can source and sink 30 mA. All I/O pins have built-in, configurable pull-up resistors. RB and RC input pins are selectable as Schmitt Trigger.

For those familiar with the Super Parallax Assembler:

The SX-Key assembler has many similarities with Parallax's Super Parallax Assembler. The similarities and differences are highlighted here.

Similarities:

- **Mnemonics:** Most mnemonic instructions are the same as in the Super Parallax Assembler. Most code already developed for PIC16C5x devices will load and assemble in the SX-Key.
- **Language Syntax:** The SX-Key Assembler uses a similar syntax as that of the Super Parallax Assembler.
- **Directives and Equates:** Directives are similar and backward compatible equates exist within the SX-Key Assembler.

Differences:

- **Environment:** The SX-Key Assembler is an integrated editor, programmer, emulator and debugger interface that runs in Windows 95/98/NT4.
- **New Mnemonics:** Several new mnemonics are available to support new opcodes in the SX chips. The new mnemonics are: 1) RET, 2) RETP, 3) RETI, 4) RETIW, 5) PAGE, 6) BANK, 7) IREAD, 8) MOV M,W, 9) MOV W,M and 10) MOV M,#lit. See Appendix B for more information.
- **Language Syntax:** Some language syntax has changed, although many of the older formats are supported for backward compatibility. For example, it is suggested that you use the \$xxxx format, instead of 0xxxxh, to specify hexadecimal numbers.
- **Equates:** Many new dynamic equates are provided for ease of use of the SX microcontroller.
- **@ sign:** The @ sign may be used in all address references to indicate an absolute, flat-architecture, address. The

What's New

assembler will handle page swapping for you by inserting the PAGE instruction before your address reference.

- **Device Settings:** All device setting symbols have been updated to describe the SX device more thoroughly. The settings that affect binary fuses indicate an “option active” state. Leaving the setting out of the device line indicates the option is inactive. See Chapter 6 for more information.

5: Introduction to the SX-Key Interface

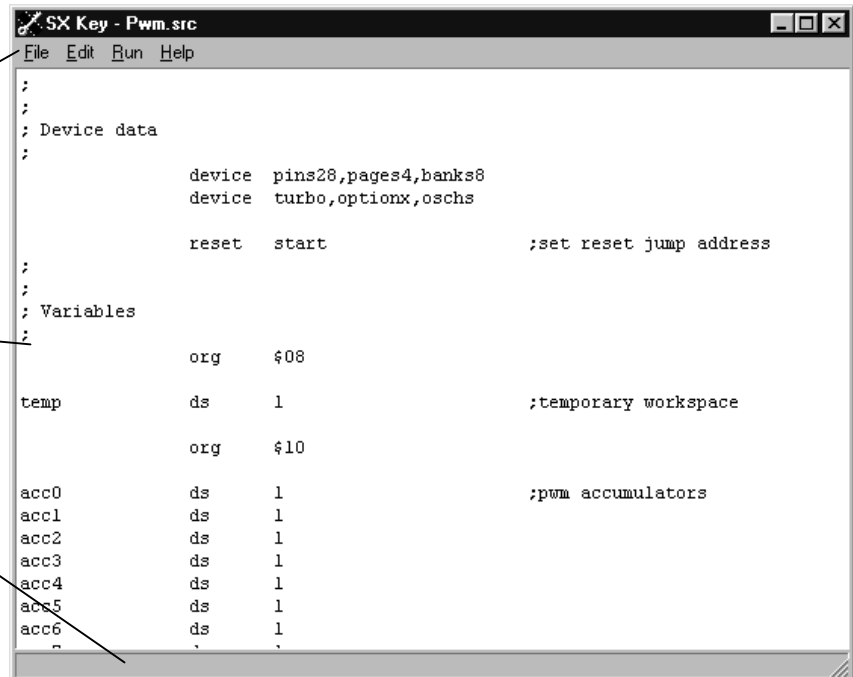
The SX-Key interface is an integrated editor, programmer, and debugger. All the functions of the SX-Key are available through this single software interface. Parallax, Inc. has made this interface as simple as possible to ease its use and to help eliminate possible confusion.

Figure 5.1: The SX-Key integrated editor, programmer and debugger.

Menus

Editor window

Status bar



Throughout the rest of this manual, the SX-Key interface will be referred to as the SX editor, or more simply, the editor.

The SX editor (see Figure 5.1, above) consists of one window containing a small menu at the top and a large editor window below. The editor window is where your SX source code may be entered and edited. Standard Windows editing shortcut keys (listed in Table 5.1, below) may be used in addition to commands in the Edit menu to manipulate the source code.

Introduction to the SX-Key Interface

Function Name	Shortcut Keys	Function Description
Copy	Ctrl-C	Copies selected text to the clipboard.
Cut	Ctrl-X	Cuts selected text to the clipboard.
Paste	Ctrl-V	Pastes clipboard contents.
Page Up	Pgup	Moves editor window one page up.
Page Down	Pgdn	Move editor window one page down.
Tab	Tab	Moves cursor to the next tab position. One tab position is set for every 8 characters.
Text Bigger	Ctrl-CrsrUp	Increase size of editor text.
Text Smaller	Ctrl-CrsrDn	Decrease size of editor text.

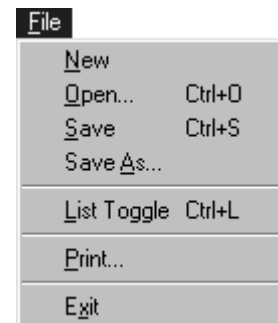
Table 5.1: Keyboard shortcuts for common editing commands.

The Menus

The SX editor menu bar contains four menus: File, Edit, Run and Help. These menus and their associated menu items are each defined below.

The File menu:

- New:** removes any text in the editor window. Use this item to start a new source code editing session.
- Open:** opens a browse window to locate and load source code files.
- Save:** saves the current source code file.
- Save As:** opens a Save As dialog box to save the current source code with a designated name.
- List Toggle:** Toggles the editor window between source code view and list view.
- Print:** opens a print dialog box to print the current source code.
- Exit:** terminates the SX-Key editor.



5: Introduction to the SX-Key Interface

Edit

Cut	Ctrl+X
Copy	Ctrl+C
Paste	Ctrl+V
Find/Replace...	Ctrl+F
Find	F3
Replace	F4
Text Bigger	Ctrl+Up
Text Smaller	Ctrl+Down

The **Edit** menu:

- Cut:** cuts the selected text from the editor window and stores it in the clipboard.
- Copy:** copies the selected text from the editor window and stores it in the clipboard.
- Paste:** pastes the text from the clipboard into the editor window starting at the current cursor location.
- Find/Replace:** opens the Find and Replace dialog box.
- Find:** finds the next occurrence of desired text.
- Replace:** replaces the selected text with desired text.
- Text Bigger:** makes editor text larger.
- Text Smaller:** makes editor text smaller.

Run

Assemble	Ctrl+A
Program	Ctrl+P
Run	Ctrl+R
Debug	Ctrl+D
Debug (reenter)	Ctrl+Alt+D
Clock...	Ctrl+K
Device...	Ctrl+I
Configure...	Ctrl+G

The **Run** menu:

- Assemble:** assembles the code.
- Program:** assembles the source code and programs the SX microcontroller (if assembly was successful).
- Run:** assembles the source code, programs the SX and generates a clock signal.
- Debug:** assembles the source code, programs the SX, generates a clock signal and initiates debug mode.
- Debug (reenter):** assembles the source code, assumes device already programmed with **Debug**, generates a clock signal and enters debug mode.
- Clock...:** opens the clock control dialog box to allow modification of the clock activity and frequency.
- Device...:** opens the device dialog box to allow modification of the SX microcontroller parameters.
- Configure...:** opens the configuration dialog box to allow modification of the SX-Key programming parameters.

Introduction to the SX-Key Interface

The Help menu:

Contents: displays symbolic debugging information.
About: displays the SX-Key Development System information box.



The Windows

Many menu items open up a separate window for further configuration or monitoring. These windows are described below.

Print Window

The Print window is accessed via the Print... item on the File menu. This window allows configuration of code for printing.

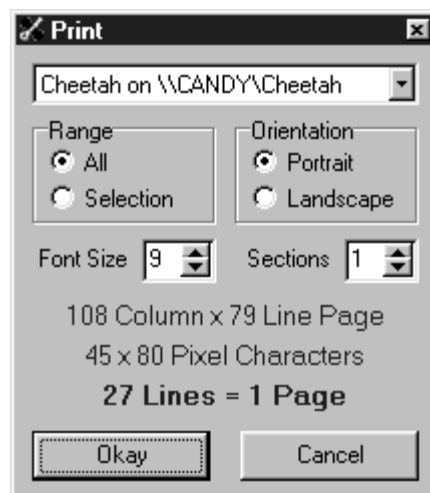


Figure 5.2: The Print window.

With the Print window, a printer, print range, orientation, font size and number of sections can be selected. The number of sections determines the number of vertical sections of code that will appear on the printed page. This allows multiple pages of code to fit on the same page.

5: Introduction to the SX-Key Interface

Find/Replace Window

The Find / Replace window is accessed via the Find / Replace... item on the Edit menu. This window allows searching for and optionally replacing text within the source code.

Figure 5.3: The Find/Replace window.



Enter the text to search for in the Find field and, if necessary, the text to replace it with in the Replace field. Select From Start to search from the start of code (deselecting this item searches from current cursor position) and Word Only to match entire words only (deselecting this item causes matching full words and portions of words).

Clicking the Find button finds the next occurrence of desired text. Clicking the Replace button replaces the selected text with the desired text. Clicking All finds and replaces all occurrences of desired text.

Once initiated, subsequent “finds” can be invoked by pressing the F3 key and “replaces” by pressing the F4 key.

Debug Windows

The Debug windows (Registers, Debug, Code, and Watch) are accessed via the Debug and Debug (reenter) items on the Run menu, or by pressing Ctrl-D or Ctrl-Alt-D. These windows display the status of the SX chip and are the interactive control panels for emulation and debugging.

Introduction to the SX-Key Interface

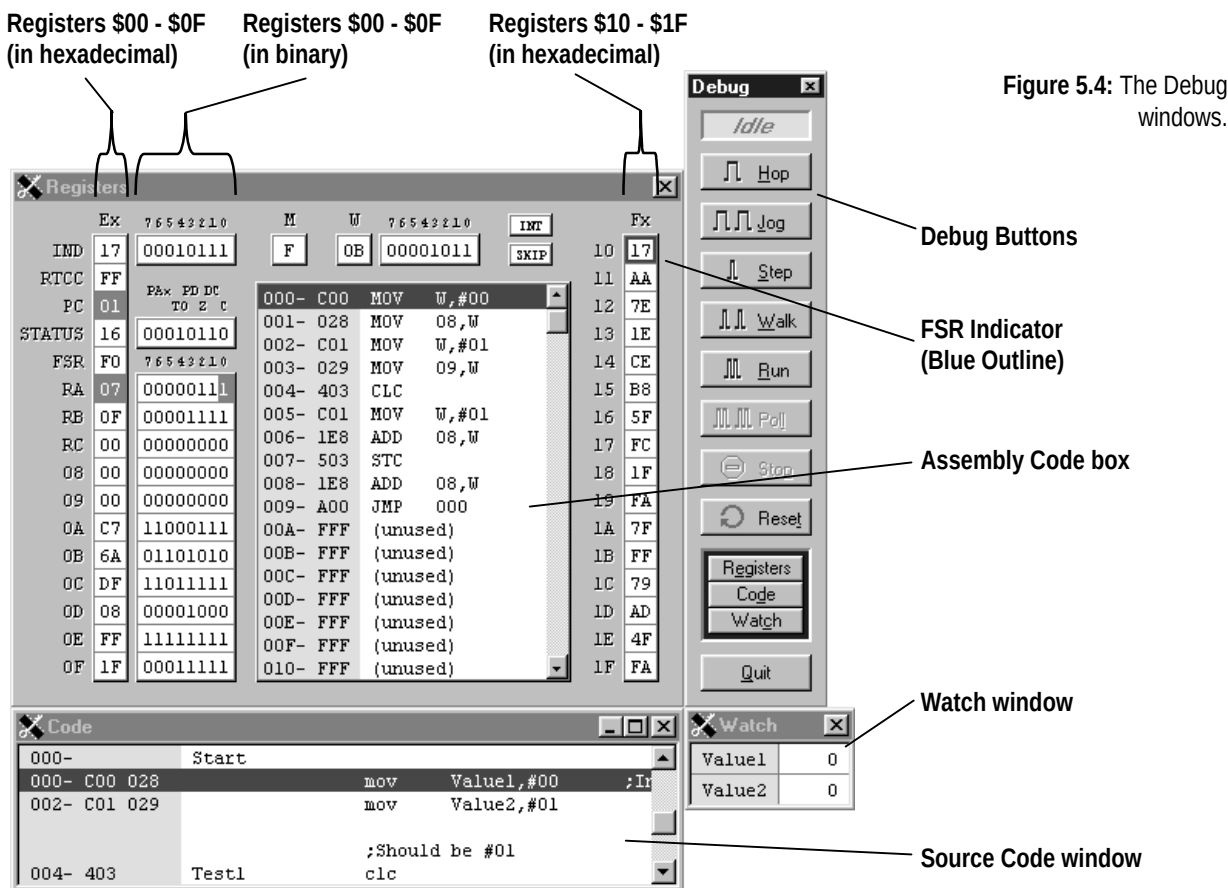


Figure 5.4: The Debug windows.

The Registers window contains all the data and describes the current state of the SX chip. The leftmost column within the Registers window displays the hexadecimal contents of registers \$00 through \$0F in the currently selected RAM bank. Registers \$10 through \$1F of all other banks are shown in the columns on the far right of the window. The bank offsets are labeled at the top of each column and the current bank is highlighted in white. These columns will expand or contract to fit the number of RAM banks available in the SX. In this example, one bank is shown.

5: Introduction to the SX-Key Interface

The blue outline, shown here on the \$10 register, indicates the location that the FSR (file select register) is currently pointing at.

The second column, just to the right of the first sixteen registers, displays a binary representation of some of the registers, namely IND, Status, RA, RB, RC and 08 through 0F.

At the top center of the Debug window is the contents of the M register (hexadecimal) and the W register (both hexadecimal and binary) and the Interrupt and Skip flags. The Interrupt and Skip flags turn blue when set and white when cleared.

The assembly code box lists several contiguous instructions at once. The first three digits on each line is the hexadecimal address, followed by the opcode and finally the assembly mnemonic and operand(s). This window normally shows the active section of code, around which the program counter (PC) points, however, the scroll bar allows movement of the window's field of view to any section of code.

The Debug window contains buttons for debug functions. Each button has an associated shortcut key, as described in Table 5.2, below.

- The Hop button will execute one source code instruction (may be multiple machine instructions) and update all registers.
- The Jog button will quickly execute one source code instruction after another, updating the display automatically and continuing until the Stop button is pressed, or a breakpoint is encountered.
- The Step button will execute one machine instruction and update all registers.
- The Walk button will quickly execute one machine instruction after another, updating the display automatically and continuing until the Stop button is pressed, or a breakpoint is encountered.
- The Run button will initiate a full-speed execution of the program, and will continue until a breakpoint is hit or the Poll or Stop buttons are pressed. The displayed registers will not update until the Poll button is pressed or execution is stopped.
- The Poll button operates in one of two modes. If a breakpoint exists, the Poll button runs the code at full speed halting execution

Introduction to the SX-Key Interface

at the break just long enough to update the display and then continues running. If a breakpoint is not set, the Poll button can be pressed only during run mode to get an instant update of the register displays.

- The Stop button halts execution of a jog, walk, run or poll operation and updates the display.
- The Reset button returns the SX chip to its initial state and sets the program counter (PC) to the location containing the reset vector.
- The Registers, Code and Watch buttons bring the associated window into view if they were hidden. (The Debug window is always on top).
- The Quit button closes the Debug windows and exits debug mode.

Button	Shortcut Keys	Function
<u>H</u> op	Alt-H	Executes one source code instruction.
<u>J</u> og	Alt-J	Executes multiple source instructions, one at a time.
<u>S</u> tep	Alt-S	Executes one machine instruction.
<u>W</u> alk	Alt-W	Executes multiple machine instructions, one at a time.
<u>R</u> un	Alt-R	Executes instructions in full speed.
<u>P</u> oll	Alt-L	Updates display then continues execution. Can be used synchronously or asynchronously.
<u>S</u> top	Alt-P	Halts execution of a walk or run operation.
<u>R</u> eset	Alt-T	Resets the SX chip.
<u>R</u> egisters	Alt-E	Brings Registers window into view.
<u>C</u> ode	Alt-D	Brings Code window into view.
<u>W</u> atch	Alt-C	Brings Watch window into view.
<u>Q</u> uit	Alt-Q	Closes the Debug Windows and exits debug mode.

Table 5.2: Summary of debug buttons and shortcut keys.

The Debug windows are highly active and interactive displays. Every time the display is updated (after a hop, jog, step, walk, run or poll operation), each register that was modified since the previous update is highlighted in red. This provides a clear indication of what the last instruction accomplished. Similarly, each bit that was changed is marked in red within all registers shown in binary. Additionally, the assembly code box and Code window highlights the instruction pointed to by the program counter (PC) in blue, and a breakpoint in red.

Color Coded Display

5: Introduction to the SX-Key Interface

Modifying registers during debugging

Any register, bits within registers, or flags can be modified using the mouse and keyboard (see Table 5.3 for a summary of the editing keys). To modify a register (in hexadecimal), first click on it or use the tab and cursor keys to move the focus to that register. (The focus is indicated by a blinking, black highlight within the register). Next, type in the new hexadecimal value on the keyboard and press the enter, space, backspace or arrow keys to write the value to the register. The new value will appear in the selected register, highlighted in red to indicate a change.

To change a bit or flag in the binary registers, simply click the mouse on the appropriate bit. The bit will toggle to the opposite state and will be highlighted in red to indicate a change. Click on the INT or SKIP flags to toggle their state. The INT and SKIP flags, unlike registers, do not indicate a change with a red highlight. Instead, a blue color indicates the flag is set, while a white color indicates the flag is cleared.

If a register's contents are changed by accident, press the ESC (escape) key to restore its previous value.

The Watch window displays the contents of selected registers in a user-defined format. The values in the Watch window can be modified using the same methods described above. In addition, the numerical values in the Watch window can be modified in any format (binary, hexadecimal or decimal) regardless of the displayed format. Simply precede the input value with a %, \$, or nothing, respectively. String values can only be modified by entering new strings. See the Watch Directive section in Chapter 6 for information of defining watches.

Table 5.3: Debug window navigation and editing keys.

Key	Function
TAB	Move focus to new control block and ignore any changes to previous register.
Cursor Keys	Move focus to new register within control and write any changes to previous register.
Space	Same as cursor down.
Backspace	Same as cursor up.
Enter	Write changes to register.
ESC	Changes register value to previous value, if it has been changed by the user. This will not work if the enter, space, backspace or cursor keys have been pressed first.

Introduction to the SX-Key Interface

The assembly code box and Code window displays a breakpoint as a red highlighted line and the next instruction to be executed as a blue highlighted line.

The breakpoint can be set to a new line by clicking the mouse button once on the desired line. Clicking the mouse button again will remove the breakpoint. Only one breakpoint can be set at a time.

The next instruction to execute can be set to a new line by double-clicking the mouse button on the desired line. Additionally, the program counter (PC) register's contents can be manually modified via the keyboard to set the next instruction to execute.

If a breakpoint and the program counter should both be on the same line, it will become multicolored. The first third of the line will be highlighted in red (to indicate the breakpoint) and the last two thirds of the line will be highlighted in blue (to indicate the next line to execute).

Debugging:

The following is required to use the debug features:

- SX-Key Rev. E (or greater),
- SX chip version 2.4 (or greater),
- No external clock source connected to the SX chip. This includes oscillator packs, crystals, resonators and RC circuits.

Source code to be debugged must include the RESET directive (see Chapter 6), must have WATCHDOG set to off, and must have 2 free words in the first page of code and 136 free words near the end of the last page of code (from 177 to 1FE, 377 to 3FE, 577 to 5FE, or 777 to 7FE), depending on the number of E²Flash pages. If an oscillator frequency of other than 50 MHz (default) is desired, the source code should contain a FREQ directive (see Chapter 6) stating the frequency.

Breakpoints and the current instruction

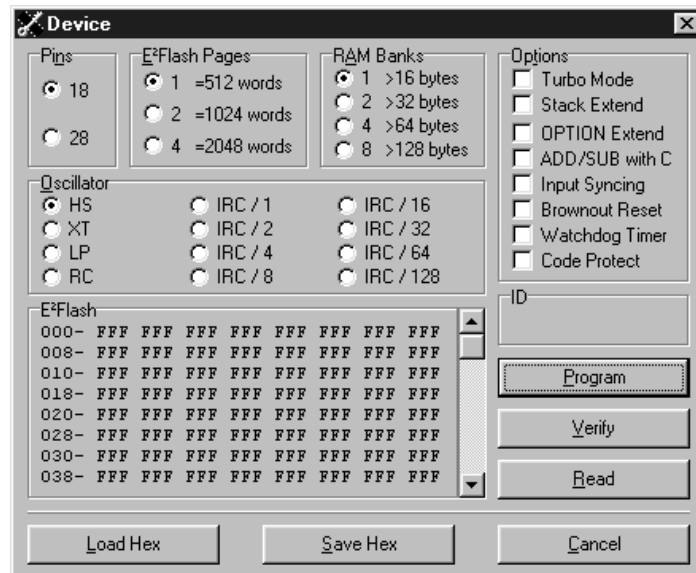
Setting the Program Counter

5: Introduction to the SX-Key Interface

Device Window:

The Device window is accessed via the Device... item on the Run menu, or by pressing Ctrl-I. This window allows you to specify all the device settings for the SX microcontroller you are using, program and verify or read the contents of the device and load or save object files for the SX.

Figure 5.5: The Device window.



All data shown in this window reflect the settings specified in the DEVICE line(s) of the source code, the settings in the loaded object code or the settings read out of the device itself. You may modify these settings manually and program or reprogram the chip, however, those modifications will not be reflected in the source code.

The Pins, section should be set to the physical pin count of the SX microcontroller being programmed. The E²Flash Pages and RAM Banks sections may be set to less than the chip's capacity in order to properly run PIC object code for lesser memory.

Introduction to the SX-Key Interface

The Options and Oscillator sections specify the configuration of the SX fuses for options like turbo mode, code protect, oscillator type and brown-out reset detection.

The ID and E²Flash sections display the values contained in the ID and E²Flash memory (the program memory) respectively.

The Program button initiates programming the SX chip with the assembled source code, or the object code loaded into the Device window.

Programming

Use the Verify and Read buttons to verify the code in the SX against that shown in the Device window or to simply read the SX's code into the Device window. These options are valuable should the code in an SX chip be questionable or unknown. Note that verifies will fail and reads will not reveal the true code if the SX chip was programmed with the code-protect fuse on. The ID field will always read properly, however.

Reading and Verifying

The Load Hex and Save Hex buttons may be used to load or save assembled object files. If an object file is desired for a particular source program, simply load the source into the SX-Key editor, assemble it, open the Device window and press the Save Hex button. To program SX chips with an object file, use the Load Hex button on the Device window to load that file and then press the Program button.

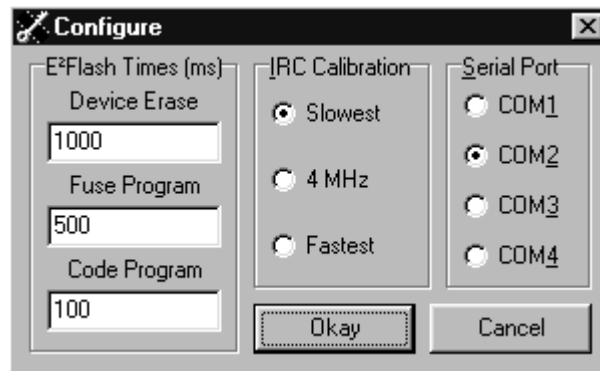
Loading and Saving Hex Files

Configure Window:

The Configure window is accessed via the Configure... item on the Run menu. This window allows modification of the parameters guiding the programming functions of the SX-Key.

5: Introduction to the SX-Key Interface

Figure 5.6: The Configure window.



You may specify which serial port the SX-Key is connected to in the Serial Port section.

The Device Erase parameter specifies how long, in milliseconds, the SX-Key should perform the erase cycle on the SX chip.

The Fuse Program and Code Program parameters specify how long, in milliseconds, the SX-Key should perform the program cycle on each fuse and EEPROM location, respectively.

All time values can be specified up to 5000 ms (5 seconds).

IRC Calibration

The IRC Calibration section allows setting the special trim feature of the SX-Key. The internal oscillator in the SX chip is based on an RC circuit. Unfortunately, RC circuits suffer timing variances due to component tolerances, temperature and other factors. Because of this, the 4 MHz internal oscillator in the SX chip may vary from frequencies way below the rated 4 MHz to frequencies way above it. This nature may make it difficult to use the internal oscillator for anything other than mundane timing activities.

The SX chip has 3 trim bits to help tune the internal oscillator and the SX-Key can automatically tune it during programming. Selecting 4 MHz in the IRC Calibration section will cause the SX-Key to tune the internal oscillator of every SX chip, during programming, to a value as close as possible to 4 MHz. The actual frequency chosen is displayed after calibration. Selecting the Slowest or Fastest options simply results in the SX-Key setting the trim bits to the slowest or fastest internal

Introduction to the SX-Key Interface

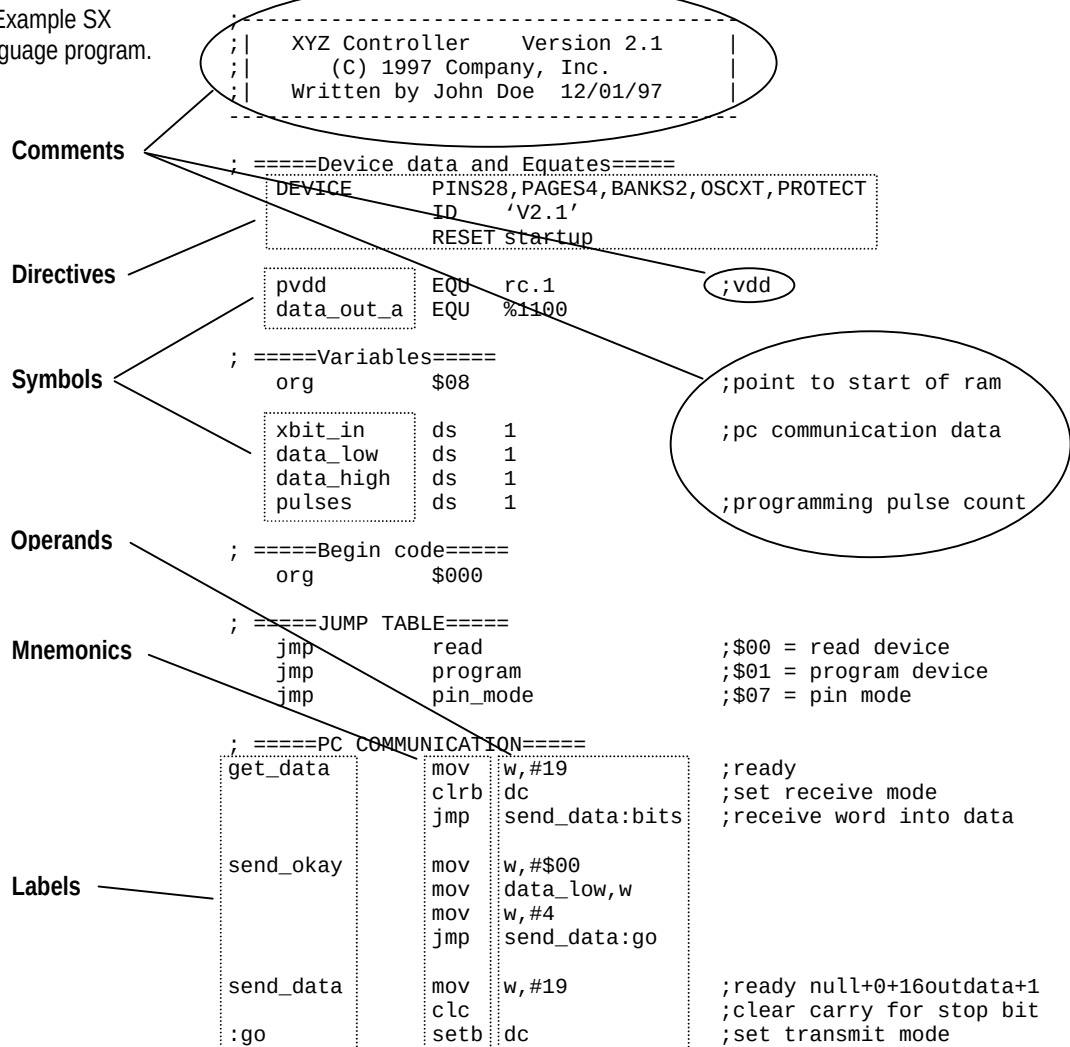
oscillator operation possible. Note that the execution speed of the SX program (when using the internal oscillator) is determined by the base oscillation speed, the divide-by ratio (IRCDIV1, IRCDIV2, etc) and the state of the Turbo fuse.

If not using the internal oscillator, select the Slowest or Fastest options from the IRC Calibration box to speed up the programming process.

6: SX-Key Assembler

A typical SX assembly language program (see Listing 6.1) contains comments, directives, symbols, labels, expressions and mnemonic instructions. This chapter discusses these source code components and demonstrates their uses.

Listing 6.1: Example SX assembly language program.



SX-Key Assembler

Comments

Comments are optional messages usually used to document the source code. They are ignored by the assembler and may be placed almost anywhere in the program. A comment must be preceded by a semicolon (;). The following demonstrates two examples of comments.

```
    ;this program controls the GPX513v driver chip
    mov     counter, 120      ;initialize loop counter
```

Notice that a comment can be placed on the same line as an instruction (see the second line above). Since the assembler ignores everything that appears to the right of a semicolon, a comment may only appear on its own line or to the right of an instruction.

For debugging purposes, lines of code can be hidden from the assembler, or commented out, simply by inserting a semicolon before the first character of the line.

Assembler Directives

Assembler directives are special instructions to the assembler to help define symbols, set device options or indicate how the code should be assembled. Directives may appear within the source code, like assembly instructions, however, since they are instructions to the assembler and not the SX microcontroller, they are not actually assembled into the final machine language program. Table 6.1 describes available SX-Key Assembler directives.

6: SX-Key Assembler

Table 6.1: SX-Key Assembler Directives.

Directive	Description	Syntax
DEVICE	Sets SX device options. These directives must precede all other directives and instructions.	DEVICE setting {,setting...}
FREQ	Specifies the clock frequency to be generated by the SX-Key during debugging.	FREQ n
ID	Assigns a value to the 8-byte ID word in the SX. The text argument may be up to 8 characters and should be in apostrophes.	ID 'text'
ORG	Specifies the starting RAM or EEPROM location of the code that follows.	ORG value
RESET	Specifies the starting location of the program in the SX's memory.	RESET label
EQU	Assigns a value to a symbol.	symbol EQU value
=	Assigns or reassigns a value to a symbol. This directive is used to create and manipulate assemble-time variables.	symbol = value
DS	Increments the memory pointer (\$) by value. Used to reserve RAM and E'Flash during assembly.	symbol DS value
DW	Defines words in EEPROM.	DW data {,data...}
IF {ELSE} ENDIF	Conditional assembly and alternate conditional assembly block.	IF condition {ELSE} ENDIF
IFDEF/IFNDEF {ELSE} ENDIF	Conditional assembly and alternate conditional assembly block based upon symbol definitions.	IF{N}DEF symbol {ELSE} ENDIF
REPT ENDR	Repeat block of code a specified number of times.	REPT count ENDR
MACRO {EXITM} ENDM	Defines a macro.	label MACRO {value} {EXITM} ENDM
EXPAND / NOEXPAND	Specifies that the following macro instructions should be, or should not be, expanded in the list file.	{NO}EXPAND
CASE / NOCASE	Specifies that the following instructions should be, or should not be, case sensitive.	{NO}CASE
BREAK	Defines a run-time breakpoint.	BREAK
WATCH	Defines a symbol to watch during debugging.	WATCH addr, count, format
ERROR	Defines an assemble-time error.	ERROR 'error text'
END	Marks the end of the source code	END

SX-Key Assembler

The device directive is perhaps the most important directive to appear in source code. The device directive specifies the number of pins, EEPROM pages, RAM banks, oscillator type and more. The various symbols for specifying these options are listed in Table 6.2.

Device Directive

Setting	Description	Default
PINS18 PINS28	Specifies the number of pins of the device.	PINS18
PAGES1 PAGES2 PAGES4	Specifies the number of EEPROM pages.	PAGES1
BANKS1 BANKS2 BANKS4 BANKS8	Specifies the number of RAM banks.	BANKS1
OSCHS OSCXT OSCLP OSCRS IRCDIV1 IRCDIV2 IRCDIV4 IRCDIV8 IRCDIV16 IRCDIV32 IRCDIV64 IRCDIV128	Specifies external crystal / resonator Specifies external crystal / resonator Specifies external crystal / resonator Specifies external RC circuit Specifies internal oscillator divided by 1 Specifies internal oscillator divided by 2 Specifies internal oscillator divided by 4 Specifies internal oscillator divided by 8 Specifies internal oscillator divided by 16 Specifies internal oscillator divided by 32 Specifies internal oscillator divided by 64 Specifies internal oscillator divided by 128	OSCHS
BROWNOUT	Specifies brown-out to trigger at 4.2 volts	no brownout
TURBO	Specifies turbo mode (1:1 execution)	compatible mode (1:4 execution)
STACKX	Stack is extended to 8 levels	2 levels
OPTIONX	Option register is extended to 8 bits	6 bits
CARRYX	ADD and SUB instructions use Carry flag as input*	Carry flag ignored
SYNC	Input Syncing enabled	Input Syncing disabled
WATCHDOG	Watchdog Timer enabled	Watchdog disabled
PROTECT	Code Protect enabled	Code Protect disabled

Table 6.2: Device Directive Settings.

* Many instructions are adversely affected by the carry flag when CARRYX is specified. See Appendix B for more information.

The device directive, if supplied, must be the first directive in the code and must appear before the first instruction. Multiple device lines can be used to accommodate many parameters as long as no conflicting parameters are given. The syntax of the device directive is:

6: SX-Key Assembler

DEVICE setting {,setting...}

The following device lines tell the assembler that the SX chip to be programmed has 18 pins, 4 EEPROM pages, 8 RAM banks, will use a high speed oscillator, will initiate brown-out at 4.2 volts, runs in turbo mode, and is code protected.

DEVICE PINS18, PAGES4, BANKS8, OSCHS
DEVICE BROWNOUT, TURBO, PROTECT

If a setting is not specified, the default is assumed. In this example, the device will also be set for 2-level stack, 6-bit option register, carry bit ignored, no input synching and no watchdog timer, since the opposing settings were not specified. Similarly, if the source code contains no device directive, the defaults will be used for all possible settings. See Table 6.2 for the defaults.

For backward compatibility with the Super Parallax Assembler, a number of other symbols are accepted for device settings. Table 6.3 lists those device setting symbols. It is recommended that all code developed for use with the SX microcontroller use the new symbols as defined in Table 6.2, above.

Table 6.3: Backward Compatible Device Directive Settings.

Setting	Description	Translation	Default
PIC16C54 PIC16C54A PIC16C55 PIC16C56 PIC16C57 PIC16C58 PIC16C58A	Specifies the number of pins, pages and banks for the device.	PINS18+PAGES1+BANKS1 PINS18+PAGES1+BANKS1 PINS28+PAGES1+BANKS1 PINS18+PAGES2+BANKS1 PINS28+PAGES4+BANKS4 PINS18+PAGES4+BANKS4 PINS18+PAGES4+BANKS4	PIC16C54
HS_OSC XT_OSC LP_OSC RC_OSC	external crystal / res. external crystal / res. external crystal / res. external RC circuit	OSCHS OSCXT OSCLP OSCRC	HS_OSC
WDT_OFF WDT_ON	Watchdog Timer setting	(no equivalent) WATCHDOG	WTD_OFF
PROTECT_OFF PROTECT_ON	Code Protect setting	(no equivalent) PROTECT	PROTECT_OFF

SX-Key Assembler

The **FREQ** (frequency) directive is used to set the frequency (in Hz) of the SX-Key's internal programmable oscillator to be used during emulation. The syntax for the **FREQ** directive is:

FREQ frequency

Note that *frequency* can be any number from 400000 to 110000000. Additionally, underscore characters can be used to help make the number more readable, as in 50_000_000 which is 50 MHz.

The **ID** (identification) directive is used to write up to eight bytes of text into the ID word of the SX chip. This is used to record a version number or other unique identification for the code. This ID word can be read out of the SX chip at any time, regardless of the code protect setting. The line below will write *GPXv2.1* into the ID word:

ID 'GPXv2.1'

The **ORG** (origin) directive tells the assembler the starting location to use for the following instructions. The syntax for the **ORG** directive is:

ORG location

Note that the **ORG** directive does not dictate whether the location is in RAM or EEPROM. The assembler simply sets the location pointer as desired and the instructions or directives following the **ORG** will be processed in relation to this pointer. The **ORG** directive is used to place data and instructions at specific locations in RAM and E²Flash.

The **RESET** directive specifies the starting address of the code to be executed when a reset condition occurs. The assembler places a 'Jump to Location' instruction at the last location in memory to facilitate this. The syntax of the reset directive is:

RESET location

The *location* argument must reside within the first page memory.

The **EQU** (equate) directive defines symbols for constants. See the section entitled "Symbols" for further details.

FREQ Directive

ID Directive

ORG Directive

Reset Directive

EQU Directive

6: SX-Key Assembler

= Directive

The = (equal) directive defines symbols for constants or assemble-time variables. This directive is similar to EQU except that any symbols created with the = directive can be reassigned new values during assemble-time with additional = directives. See the section entitled “Symbols” for further details.

DS Directive

The DS (define space) directive increments the location pointer during assembly. This may be used to cause the assembler to arrange sequential RAM assignments (arrays). For example:

```
ORG      $08
Array    DS   3
Other    DS   2
```

defines 3 bytes, starting at location 8, for the *Array* symbol. *Array+1* is the second element of the array and *array+2* is the third element. The *Other* symbol’s first and second elements are placed starting with location \$0B. Note that no code is generated in the example above.

DW Directive

The DW (define word) directive defines 12-bit words in EEPROM. This is used to store a data table in the SX EEPROM space. For example:

```
DW      $FFF, $009, $1A0
```

stores the values \$FFF, \$009 and \$1A0 into EEPROM starting at the current location. See Chapter 7 for more information on using tables.

IF..ELSE Directive

The IF...ELSE directive is used to create conditional assembly blocks. A conditional assembly block is source code that is assembled only if the specified condition is true; otherwise, the code block is ignored by the assembler. Conditional assembly allows for easy code customization for multiple applications. For example, it might be necessary to produce a number of related products all based upon the same main source code but each having a small portion of unique code. The syntax for the IF...ELSE directive is:

```
IF      condition
        codeblock
{ELSE
        codeblock}
ENDIF
```

SX-Key Assembler

Note that the ELSE block is optional and the ENDIF is required to end the conditional block. The comparison operators for the condition argument are listed in Table 6.4 below.

Symbol	Operation
=	Equal
<>	Not equal
<	Less than
>	Greater than
=<	Equal or less than
=>	Equal or greater than

Table 6.4: Comparison Operators.

The following example demonstrates the use of the IF...ELSE directive.

```
Delay EQU 10
Mode EQU 1

IF Delay >=9
    mov $08, #5
    add $09, #%011
ENDIF

IF Mode = 0
    mov $0A, #$1B
ELSE
    mov $0A, #$1C
ENDIF
```

The condition for an IF...ELSE directive can contain expressions and multiple conditional statements. Two or more conditional statements may be specified by appending them together with the conditional operators NOT, AND, OR and XOR. For example:

```
IF mode=1 AND delay>9
```

would assemble the code block following it if *mode* equals 1 and *delay* is greater than 9 at assemble time. If the statement or expression evaluates to anything other than zero (0), the condition is true; otherwise, the condition is false.

The IFDEF...ELSE (if defined) and IFNDEF...ELSE (if not defined) directives are very similar to the IF...ELSE directive. The difference is

IF{N}DEF..ELSE Directive

6: SX-Key Assembler

they assemble or prevent assembly of code blocks based on whether a symbol is defined or not. For example:

```
DriverOn    EQU    1

IFDEF  DriverOn
{some code block here}
ENDIF
```

would assemble the code block in the IFDEF statement because the *DriverOn* symbol was defined. If *DriverOn* was not defined (ie: line number 1, above, was commented out) the code block would be ignored.

Repeat Directive

The REPT (repeat) directive is used to indicate that a block of code is to be repeated a specified number of times during assembly. The syntax for the REPT directive is:

```
REPT    count
codeblock
ENDR
```

Note that *count* must be greater than 0 and ENDR is required to end the repeat block. For example:

```
REPT    3
add     $0A, #$01
ENDR
```

would result in the source code being expanded to:

```
add     $0A, #$01
add     $0A, #$01
add     $0A, #$01
```

during the assembly of the code.

Within a repeat block, the percent sign (%) alone may be used to refer to the current iteration (1–n) of the block during assembly. For example:

SX-Key Assembler

```
REPT 3
add $0A,#%
ENDR
```

would result in:

```
add $0A,#1
add $0A,#2
add $0A,#3
```

during the assembly of the code. In other words, during assembly, the first time through the repeat block, the % symbol is equal to 1, the second time through it is equal to 2, etc.

The MACRO directive defines blocks of code that can be reused in multiple places. This is helpful when source code contains numerous sections with similar routines. A macro allows the definition of this routine only once and it can be added to code in multiple places simply by referencing its name. The syntax for the macro directive is:

Macro Directive

```
label      MACRO      {argcount}
            codeblock
            {EXITM}
            ENDM
```

Note that *argcount* and EXITM are optional and ENDM is required to end the macro block. *Argcount*, when provided, specifies the exact number of arguments required by the macro and must be in the range 0 to 64. An error will occur during assembly if the macro is invoked with more or less than the required number of arguments. If the *argcount* argument is left out, the macro can accept any number of arguments. To invoke a macro within the code, simply reference it's name followed by the desired arguments. The following example defines a macro, called Add15, which adds the argument + 15 to registers \$08 through \$0A.

```
Add15      MACRO      1
            mov $08,#1+15
            mov $09,#1+15
            mov $0A,#1+15
            ENDM
```

6: SX-Key Assembler

Within a macro, arguments can be referenced by their index value preceded with a backslash (\), as in the code above. The first argument is \1. The second argument would be \2, etc. If the macro were called with the value 50 as it's first argument, then every occurrence of \1 within the macro would be replaced with 50 at the time of assembly. The reference \0 returns the *number* of arguments passed to the macro. This is useful if a macro has an unspecified number of valid arguments. In fact, inside the macro, a REPT block could be used to cycle through the arguments by using \%.

To use this macro, simply call it by name along with it's required argument at the place in the code where it is needed. For example:

```
add      ra,#2
jc       Overflow
Add15    21
```

would result in the code being expanded, during assembly, to the following:

```
add      ra,#2
jc       Overflow
mov      $08,#36
mov      $09,#36
mov      $0A,#36
```

If a macro call requires multiple arguments, specify them in the calling line separated by commas (,).

The expansion of macros can be controlled during list generation with the EXPAND and NOEXPAND directives, see below. The default is to expand all macros within a list file.

The optional EXITM directive can be used to prematurely end macros. This is useful inside an IF...ELSE conditional block within a macro to end the macro at various points based on certain conditions.

Macros may call other macros as well as themselves. This means recursive macro calls are possible.

SX-Key Assembler

The EXPAND and NOEXPAND directives specify how to handle macro calls for the purposes of list generation. If a list file is needed that has no macro mnemonics expanded, simply place the noexpand directive above the first macro call. The expand and noexpand directives can be used as often as desired and will only affect the code below them. For example, if source code referenced two macros, M1 and M2, and a list file was needed with only the M1 macro expanded, the expand / noexpand directives might be used as follows:

```
{macro definitions and other code appears above this point}
M1          {arguments here}

NOEXPAND
M2          {arguments here}
EXPAND

M1          {arguments here}
```

The above code will result in a list file with the first and last macro calls expanded and the second macro call unexpanded. Note that expand is the default so the expand directive was not used above the first macro call. Additionally, the list file will always show addresses and assembly within a macro regardless of the use of expand and noexpand directives.

The CASE and NOCASE directives specify how to handle the character case (upper or lower) of symbols in source code. The CASE directive will cause all the code below it, up to a NOCASE directive, to be case sensitive. The NOCASE directive will cause all code below it, up to a CASE directive, to be case insensitive. The default is case insensitive. The CASE and NOCASE directives can be used as often as desired and will only affect the code below them.

Using case sensitivity will allow symbols with the same name, but different character cases, to be treated as different symbols. For example:

```
CASE
temp   EQU $01
Temp   EQU $02
```

Expand and noexpand Directives

CASE and NOCASE Directives

6: SX-Key Assembler

The above code would assemble properly and would have two distinct symbols, *temp* and *Temp*.

NOTE: All directives, instructions and reserved words must be specified in upper case when CASE is active. Using case sensitive mode can be very tricky, can easily lead to wasted time spent debugging, and is not recommended.

Break Directive

The BREAK directive causes a breakpoint to be set at the first line of executable code immediately following it. This is used to set and save a breakpoint in the source code in order to avoid the need to manually set a breakpoint in the Debug window. The syntax of the break directive is:

```
BREAK  
breakpoint code
```

where *breakpoint code* is the desired line of source code to break on. The BREAK directive is ignored during any operation other than Debug. Only one breakpoint can be defined at a time.

Watch Directive

The WATCH directive allows the definition of format for viewing and modifying variables at runtime during debug mode. The variable's bit address, number of bits or bytes to view, and display format may be specified. The syntax for the WATCH directive is:

```
WATCH symbol{.bit}, count, format
```

The *symbol* argument can be a symbol name or register address and can optionally specify a bit address within the symbol. If no bit address is specified, bit 0 is assumed. The *count* argument indicates the number of bits (1-32) or bytes (1-16) to include in the displayed value. When using the FSTR or ZSTR format, the *count* is the number of bytes while in all other formats the *count* is the number of bits. Up to 32 WATCH directives can be specified. Table 6.5 lists available format settings for the WATCH directive.

SX-Key Assembler

Format	Operation
UDEC	Displays value in unsigned decimal format
SDEC	Displays value in signed decimal format
UHEX	Displays value in unsigned hexadecimal format
SHEX	Displays value in signed hexadecimal format
UBIN	Displays value in unsigned binary format
SBIN	Displays value in signed binary format
FSTR	Displays values in fixed-length string format
ZSTR	Displays values in zero-terminated format with maximum size

Table 6.5: Format Settings for the WATCH directive.

The WATCH directive may be specified anywhere in the source code; however, it is suggested that it be specified near the top, where symbols are defined. When code containing one or more WATCH directives is assembled and programmed using Debug mode, a Watch window appears, along with the other debugging windows, to display the results. The Watch window display is updated at the same time the Registers and Code windows are updated. Typically, WATCH directives are used in conjunction with the BREAK directive, however, asynchronous breaks and polls (with the Poll button) may be used to update the display as well.

The following code snippet is an example of multiple WATCH directives and their output:

```

LCounter    EQU    $08
Hcounter    EQU    $09
Flags       EQU    $0A
String      EQU    $0B    ;(3 chars, $0B - $0D)
WATCH       LCounter, 16, UDEC
WATCH       Flags.0, 1, UBIN
WATCH       Flags.1, 1, UBIN
WATCH       String, 3, FSTR

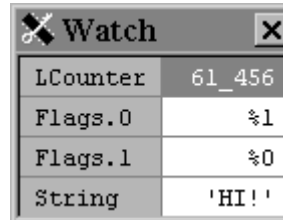
Start       ;Start of main routine
MOV         LCounter, #$10
MOV         HCounter, #$F0
MOV         Flags, #%01
MOV         String, #'H'
MOV         String+1, #'I'
MOV         String+2, #'!'

```

6: SX-Key Assembler

This code snippet would result in a Watch window similar to below:

Figure 6.1: The Watch window.



The first variable in the Watch window is created by the WATCH directive on line 5 of the source code. It tells the SX-Key to display a value starting at bit 0 of LCounter (since bit 0 is assumed if no bit address is specified) containing 16 bits and formatted as an unsigned decimal number. Note that although LCounter is only an 8-bit register, the WATCH directive can span multiple registers (from low-byte to high-byte) in order to construct up to a 32-bit value. In this case, LCounter (\$08) is the lower 8 bits and HCounter (\$09) is the upper 8 bits of the displayed value (61,456 or \$F010).

The second and third variables in the Watch window are created by the WATCH directives on lines 6 and 7 of the source code. Both are single bit values, shown in unsigned binary format. The first of these corresponds to Flags' bit 0 (Flags.0) and the second corresponds to Flags' bit 1 (Flags.1) as specified by the *symbol* arguments.

The fourth variable in the Watch window is created by the last WATCH directive. It tells the SX-Key to display a fixed-length string (FSTR) of 3 bytes starting with the String symbol. In this case, 'HI!' is displayed.

If the ZSTR format is used, the Watch window will display all bytes up to a byte equal to zero, or up to the maximum length (specified by the *count* argument).

The WATCH directive can not span register banks. If a WATCH directive indicates to use a number of bits or bytes that would take the value across a bank's boundary, such as WATCH \$1F, 16, UDEC, the WATCH directive wraps around to the first location in that bank. This situation should be avoided, as the value displayed may not be useful.

SX-Key Assembler

The values in the Watch window can be modified just like registers in the Registers window. See *Modifying Registers During Debugging* in Chapter 5 for more information.

The `ERROR` directive causes an error message to be displayed in the status bar at assemble-time. This is usually used within a conditional block to halt assembly if a certain condition is not met. The syntax of the error directive is:

```
ERROR      'errormessage'
```

The *errormessage* argument can contain static text messages only. For example, the following line, if reached during assembly, will halt assembly and display *Illegal Argument* on the screen.

```
ERROR      'Illegal Argument'
```

Error Directive

Symbols

Symbols are descriptive names for 32-bit numeric values. Symbol names can consist of up to 32 alphanumeric and underscore (`_`) characters and must start with a letter or underscore. Symbols for constants are usually defined near the start of assembly source code using the equal directive `EQU`. For example:

```
loop_count EQU 10
index      EQU $08
```

defines the symbol *loop_count* to be equal to the number 10, and the symbol *index* to be equal to the hexadecimal number 08. During assembly, everywhere the symbol *loop_count* appears in the code, the number 10 will be used. Similarly, the hexadecimal number 08 will be used in place of the symbol *index* during assembly.

Symbols are a great way to define names for registers and constants that would otherwise appear in many places of a program as just non-descriptive numbers. Assembly code that makes ample use of symbols is easier to read and debug and requires fewer comments.

6: SX-Key Assembler

The = directive also assigns values to symbols, but unlike EQU, can reassign values to those symbols. This directive is useful in macros or repeat blocks to create assemble-time variables. For example:

```
count      =    0
ORG        $20
REPT       5
            DW  count+1*2
            count = count + 3
ENDR
```

stores the values 2, 8, 14, 20 and 26 in memory locations \$20 through \$24 at assemble-time.

Labels

Labels are descriptive names that are given to sections of code. Similar to symbol names, label names can consist of up to 32 alphanumeric and underscore (_) characters and must start with a letter or underscore. Labels are used to direct code execution to specific routines, and are defined simply by specifying the label name before the routine. For example:

```
main      mov    index, #15
          jmp    main
```

defines the label *main* to point at the first line of code. Later in the program (the second line in this example) to continue execution back at the first line, the jump command is used with the argument *main*. This is usually read as, “jump to main.”

There are three types of labels: global, local, and macro. A global label is one that is unique for the entire length of the code. No two global labels can exist with the same name. A local label is one that is unique for only a portion of the code and must begin with a colon (:). A local label can have the same name as one or more other local labels in a program but they must be separated by at least one global or macro label. A macro label is the name of a macro block (see the MACRO directive, above) and is unique. No macro label can exist with the same

SX-Key Assembler

name as another macro, global or local label. The following code demonstrates global and local labels.

```
main      mov     loop_count, #15 ;initialize loop_count
:loop     mov     $09, #100       ;set some other register
          djnz    loop_count, :loop ;decrement loop_count,
                                ;jump to :loop if not zero
continue  move    loop_count, #50 ;set loop_count to 50
:loop     djnz    loop_count, :loop ;decrement loop_count,
                                ;jump to :loop if not zero
          jmp     main ;start over
```

The example above contains two global labels, *main* and *continue*, and two local labels, both named *loop*. The area between the *main* and *continue* global labels is where a local label can exist. Since the *djnz* instruction in line 3 references the *:loop* label, and line 3 is between the global labels *main* and *continue*, it will only jump to the *:loop* label at line 2 and not the *:loop* label at line 6.

Local and global labels are also allowed within a macro. It is suggested that global labels not be used within a macro, however, as that would prevent the macro from being called more than once.

Expressions

Expressions may be used within the arguments of instructions and directives to calculate values at assemble time. The use of expressions helps build more maintainable, easier to understand code. For example, if a program uses values that are all related to the same base number, it makes sense to include an expression crafted from that relationship. If a symbol *N* is defined as being equal to the base number 2, then *N*2+1* and *N*3* can be used to derive values related to it; 5 and 6 in this case. At a later time, it might become necessary to adjust the base value to 3 and since expressions were used to derive the related values, only the symbol *N* needs to be modified.

All expressions are evaluated at assemble-time only; they are never reevaluated during run-time. This means symbols for unknowns, such as registers (which change at runtime), can not be included inside expressions.

6: SX-Key Assembler

All constants and symbols within expressions may be 32-bit numbers and the result of the expression is a 32-bit number. Since the SX is only an 8-bit processor, care must be taken to extract the appropriate portion of the result for any particular operation. Table 6.5 and 6.6 describes the available unary and binary operators for use within expressions.

Table 6.5: Unary Operators.

Symbol	Operation
	Absolute Value
-	Negative
~	Not

Table 6.6: Binary Operators.

Symbol	Operation
+	Addition
-	Subtraction
*	Multiplication
/	Division
//	Modulus
&	Logical AND
	Logical OR
^	Logical XOR
<<	Shift Left
>>	Shift Right
><	Reverse bits

Expressions are evaluated strictly from left-to-right, except when parenthesis are present. The following code demonstrates some simple expression usage.

```
loop_count    equ 5*3
table         ds 5+(2*3)
mov           loop_count+10/5, #%01101
```

The second expression, above, will evaluate to 11; the expression within parentheses is evaluated first. The third expression, `loop_count+10/5`, will evaluate to 5; $15 + 10 = 25 / 5 = 5$.

SX-Key Assembler

Data Types

The SX-Key Assembler understands four data types: decimal, binary, hexadecimal and ASCII. Table 6.7 describes these data types and their syntax.

Data Type	Syntax	Example
Decimal	xxxx	1250
Binary	%xxxx	%01101010
Hexadecimal	\$xxxx	\$1AC6
ASCII	'x'	'S'

Table 6.7: Data Types.

Note: For compatibility with the Super Parallax Assembler, the notation xxxxb and xxxxh is accepted for the binary and hexadecimal data types, respectively. It is suggested that all new SX assembly code use the notation listed in Table 6.7.

Any numbers that evaluate to a result greater than 32-bits wide will cause an error at assemble time. The largest decimal number allowed is 4_294_967_295. The largest hexadecimal number allowed is \$FFFF_FFFF. The largest binary number allowed is %1111_1111_1111_1111_1111_1111_1111_1111.

7: SX Special Features and Coding Tips

Introduction

The SX chip offers many configurable features. This chapter explains how to use many of these features, offers coding tips and demonstrates most topics with sample code. Many of the code snippets can be found in the Help_Code directory of the SX-Key Development System diskette.

Port Configuration and Usage

There are many configuration options for each of the ports on the SX chip as shown in Table 7.1. The following sections explain how to use the various port configuration options.

Table 7.1: Port configuration options.

	Port A	Port B	Port C
Input/Output	X	X	X
Pull-Ups	X	X	X
CMOS/TTL	X	X	X
Schmitt-Trigger		X	X
Edge-Interrupts		X	
Comparator		Three Pins	

To set these functions, a special form of the MOV instruction, called the port configuration instruction, is used to modify the port configuration registers. The syntax of this instruction is:

MOV !port, src

By default, the port configuration instruction writes to the port direction registers, called the tristate registers. To write to other registers, the MODE register must be preset with a specific value. Table 7.2 lists these values. See Appendix F for more information.

SX Special Features and Coding Tips

MODE	Port A	Port B	Port C
\$0F	TRIS_A	TRIS_B	TRIS_C
\$0E	PLP_A	PLP_B	PLP_C
\$0D	LVL_A	LVL_B	LVL_C
\$0C		ST_B	ST_C
\$0B		WKEN_B	
\$0A		WKED_B	
\$09		Swap W with WKPEN_B	
\$08		Swap W with COMP_B	
\$07 - \$00			

Table 7.2: Mode value definitions.

Each of the I/O pins in each of the ports can be configured to an input or output direction by writing to the appropriate tristate register (TRIS_A, TRIS_B or TRIS_C). The default I/O pin direction is input. I/O pin direction configuration is usually done once, near the start of code, however, the pin directions can be changed multiple times at any place in the code.

Port Direction

To configure the direction of the I/O pins to inputs or outputs:

- 1) Set the MODE register to \$0F (the default value at startup).
- 2) Use the port configuration instruction to set the individual directions of each I/O pin within each port. A high bit (1) sets the corresponding pin to input mode and a low bit (0) sets the pin to output mode.

The following code snippet demonstrates this:

(See Help_Code\Port_Direction.src on the diskette for full example)

```
;Direction Configuration
MODE $0F ;Set Mode to allow Direction configuration
MOV !ra,##%0000 ;Port A bits 0-3 to output
MOV !rb,##%11110000 ;Port B bits 4-7 to input, bits 0-3 output
MOV !rc,##%00001111 ;Port C bits 4-7 to output, bits 0-3 input
```

7: SX Special Features and Coding Tips

If the logic-level of output pins are expected to begin at a certain state (0 or 1), care should be taken to set the output latch appropriately before setting the pin's direction to output. Failing to do so may result in a momentary glitch on the pin during initialization. For example, if all output pins were expected to begin in a low state (0), insert the following lines above the previous code snippet:

```
;Set output pins low
MOV    ra, #%0000          ;Port A bits 0-3 low
MOV    rb, #%00000000      ;Port B bits 0-7 low
MOV    rc, #%00000000      ;Port C bits 0-7 low
```

Pull-up Resistors

Every I/O pin has optional internal pull-up resistors that can be configured by writing to the appropriate pull-up register (PLP_A, PLP_B or PLP_C). By configuring pull-up resistors on input pins, the SX chip can be connected directly to open/drain circuitry without the need for external pull-up resistors. The internal pull-up resistors are disabled by default. Pull-up resistors can be activated for all pins, regardless of pin direction but really matter only when the associated pin is set to input mode.

To configure the I/O pins to have internal pull-up resistors:

- 1) Set the MODE register to \$0E (the value for pull-up register configuration).
- 2) Use the port configuration instruction to set the individual pull-up state of each I/O pin within each port. A high bit (1) disables the pull-up for the corresponding pin and a low bit (0) enables the pull-up resistor for a pin.
- 3) Set I/O pin directions as necessary.

SX Special Features and Coding Tips

The following code snippet demonstrates this:
(See Help_Code\Pull-Up.src on the diskette for full example)

```
;Pull-Up Resistor Configuration
MODE  $0E          ;Set Mode for Pull-Up Resistor config.
MOV   !ra,##%0000   ;Port A bits 0-3 to pull-ups
MOV   !rb,##%11110000 ;Port B bits 4-7 normal, bits 0-3 pull-ups
MOV   !rc,##%00001111 ;Port C bits 4-7 pull-ups, bits 0-3 normal
MODE  $0F          ;Set Mode to allow Direction configuration
MOV   !ra,##%1111   ;Port A bits 0-3 to input
MOV   !rb,##%00001111 ;Port B bits 4-7 to output, bits 0-3 input
MOV   !rc,##%11110000 ;Port C bits 4-7 to input, bits 0-3 output
```

Every I/O pin has selectable logic level control that determines the voltage threshold for a logic level 0 or 1. The default logic level for all I/O pins is TTL but can be modified by writing to the appropriate logic-level register (LVL_A, LVL_B or LVL_C). The logic level can be configured for all pins, regardless of pin direction, but really matters only when the associated pin is set to input mode. By configuring logic levels on input pins, the SX chip can be sensitive to both TTL and CMOS logic thresholds. Figure 7.1 demonstrates the difference between TTL and CMOS logic levels.

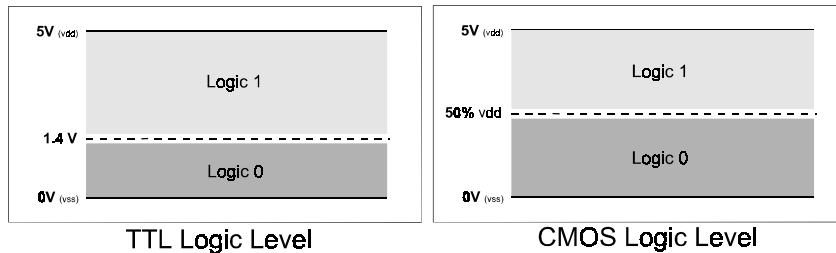


Figure 7.1: TTL and CMOS Logic Levels

The logic threshold for TTL is 1.4 volts; a voltage below 1.4 is considered to be a logic 0, while a voltage above 1.4 is considered to be a logic 1. The logic threshold for CMOS is 50% of v_{dd} , a voltage below $\frac{1}{2}v_{dd}$ is considered to be a logic 0, while a voltage above $\frac{1}{2}v_{dd}$ is considered to be a logic 1.

7: SX Special Features and Coding Tips

To configure the I/O pins to use CMOS- or TTL-level logic:

- 1) Set the MODE register to \$0D (the value for logic-level register configuration).
- 2) Use the port configuration instruction to set the individual logic-level state of each I/O pin within each port. A high bit (1) sets the corresponding pin to TTL-level logic and a low bit (0) sets it to CMOS-level logic.
- 3) Set I/O pin directions as necessary.

The following code snippet demonstrates this:

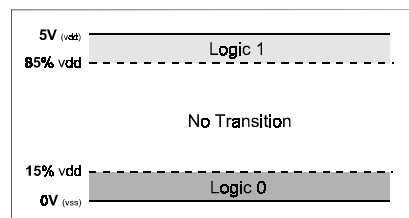
(See Help_Code\Logic_Level.src on the diskette for full example)

```
;Logic Level Configuration
MODE $0D          ;Set Mode to Logic Level configuration
MOV  !ra,#%0000    ;Port A bits 0-3 to CMOS
MOV  !rb,#%11110000 ;Port B bits 4-7 to TTL, bits 0-3 CMOS
MOV  !rc,#%00001111 ;Port C bits 4-7 to CMOS, bits 0-3 TTL
MODE $0F          ;Set Mode to allow Direction configuration
MOV  !ra,#%1100    ;Port A bits 0-1 to output, bits 2-3 input
MOV  !rb,#%10110011 ;Port B bits 2,3,6 to output, all others input
MOV  !rc,#%11011110 ;Port C bits 0,5 to output, all others input
```

Schmitt-Trigger

Every I/O pin in port B and port C can be set to normal or Schmitt-Trigger input. This can be configured by writing to the appropriate Schmitt-Trigger register (ST_B or ST_C). The I/O pins are set to normal input mode by default. Schmitt-Trigger mode can be activated for all pins, regardless of pin direction but really matter only when the associated pin is set to input mode. By configuring Schmitt-Trigger mode on input pins, the SX chip can be less sensitive to minor noise on the input pins. Figure 7.2 details the characteristics of Schmitt-Trigger inputs.

Figure 7.2: Schmitt-Trigger Input Levels.



SX Special Features and Coding Tips

Schmitt-Trigger inputs are conditioned with a large area of hysteresis. The threshold for a logic 0 is 15% of vdd and the threshold for a logic 1 is 85% of vdd. The input pin defaults to an unknown state until the initial voltage crosses a logic 0 or logic 1 boundary. A voltage must cross above 85% of vdd to be interpreted as a logic 1 and must cross below 15% of vdd to be interpreted as a logic 0. If the voltage transitions somewhere between the two thresholds, the interpreted logic state remains the same as the previous state.

To configure the I/O pins to Schmitt-Trigger input:

- 1) Set the MODE register to \$0C (the value for Schmitt-Trigger register configuration).
- 2) Use the port configuration instruction to set the individual Schmitt-Trigger state of each I/O pin within each port. A high bit (1) sets the corresponding pin to normal and a low bit (0) sets it to Schmitt-Trigger.
- 3) Set I/O pin directions as necessary.

The following code snippet demonstrates this:

(See Help_Code\Schmitt-Trigger.src on the diskette for full example)

```
;Schmitt-Trigger Configuration
MODE    $0C          ;Set Mode to Schmitt Trigger configuration
MOV     !rb,#%11110000 ;Port B bits 4-7 to normal, bits 0-3 to S.T.
MOV     !rc,#%00001111 ;Port C bits 4-7 to S.T., bits 0-3 normal
MODE    $0F          ;Set Mode to allow Direction configuration
MOV     !rb,#%10110011 ;Port B bits 2,3,6 to output, all others input
MOV     !rc,#%11011110 ;Port C bits 0,5 to output, all others input
```

Every I/O pin in port B can be set to detect logic level transitions (rising edge or falling edge). This can be configured by writing to the Edge Selection register (WKED_B) and detected by monitoring the Pending register (WKPEN_B). The I/O pins are set to detect falling edge transitions by default. By configuring edge detection on input pins, the SX chip can set the pin's associated bit in the Pending register when the desired edge arrives. The Pending register bits will never be cleared by the SX alone; the running program is responsible for doing so. This means, if a desired edge is detected, the flag indicating this will remain

Edge Detection

7: SX Special Features and Coding Tips

set until the program has time to attend to it. This feature can be used by the SX program for signals that need attention, but not necessarily immediately.

To configure the I/O pins for edge detection:

- 1) Set the MODE register to \$0A (the value for Edge Detect register configuration).
- 2) Use the port configuration instruction to set the individual edge to detect on each I/O pin. A high bit (1) sets the corresponding pin to falling-edge detection and a low bit (0) sets it to rising-edge detection.
- 3) Set I/O pin directions as necessary.

The following code snippet demonstrates this:

(See Help_Code\Edge_Detection.src on the diskette for full example)

```
                                ;Edge Detection Configuration
Start
MODE  $0A                      ;Set Mode to allow Edge configuration
MOV   !rb,##%11111111         ;Port B bits 0-7 to falling edge
MODE  $0F                      ;Set Mode to allow Direction configuration
MOV   !rb,##%11111111         ;Port B bits 0-7 to input

Main
MODE  $09                      ;Set Mode for Pending register check
MOV   !rb,##%00000000         ;This line moves WKPND_B to W and
                                ;writes all zeros to WKPND_B
JMP   Main                    ;At this point, W should be checked for
                                ;high bits, indicating a falling edge occurred
```

SX Special Features and Coding Tips

The following are points to remember with edge detection:

- The edge detection feature is always enabled and the Pending register is always updated even if the SX program does not configure or use it.
- It is up to the SX program to clear the bits of the Pending register when detection of a future transition is desired. The `MOV !rb,#%00000000` instruction effectively clears all bits of the Pending register at the same time that it stores the current edge detection status in W.
- If the SX program is designed to handle only one edge detection event at a time (on two or more pins), it will be necessary to get the status (as shown above), clear only the bit being attended to and move the modified status back to the Pending register.
- An edge detection event will not wake up the SX chip from a SLEEP mode unless the Wake-Up Enable mode is also set. See below for more information.

Every I/O pin in port B can be set to cause an interrupt upon logic level transitions (rising edge or falling edge). By configuring interrupts on input pins, the SX chip can respond to signal changes in a quick and deterministic fashion. In addition, an interrupt of this sort will wake up the SX chip from a SLEEP state. This can be configured by writing to the Edge Selection register (WKED_B) and the Wake-Up Enable register (WKEN_B) and detected by monitoring the Pending register (WKPEN_B) in the interrupt routine. The I/O pins have interrupts disabled and are set to detect falling edge transitions by default.

As with edge selection, the Pending register bits will never be cleared by the SX alone; the running program is responsible for doing so. This means if a desired edge is detected, the interrupt will occur and the flag indicating this will remain set until the program clears it. Additional transitions on that pin will not cause interrupts until the associated bit in the Pending register is cleared.

Wake-Up (Interrupt) on Edge Detection

7: SX Special Features and Coding Tips

To configure the I/O pins for wake-up (interrupt) edge detection:

- 1) Set I/O pin edge detection as desired. (See Edge Detection, above, for more information).
- 2) Set the MODE register to \$0B (the value for Wake-Up Enable register configuration).
- 3) Use the port configuration instruction to enable the individual pins for wake-up interrupts. A high bit (1) disables interrupts and a low bit (0) enables interrupts.
- 4) Set I/O pin directions as necessary.
- 5) Clear the Pending register to enable new interrupts.

The following code snippet demonstrates this:

(See Help_Code\Wake-Up_Enable.src on the diskette for full example)

```
RESET Start

Interrupt                                ;Interrupt routine (must be at address $0)
ORG $0
MODE $09                                ;Set Mode for Pending register
MOV !rb,%00000000                       ;Clear Pending/get current status in W
RETI                                     ;rest of interrupt routine goes here

;Wake-Up Edge Detection Configuration

Start
MODE $0A                                ;Set Mode to allow Edge configuration
MOV !rb,#%11111111                      ;Port B bits 0-7 to falling edge
MODE $0B                                ;Set Mode to allow Wake-Up configuration
MOV !rb,%11110000                       ;Port B bits 4-7 to normal, 0-3 to Wake-Up
MODE $0F                                ;Set Mode to allow Direction configuration
MOV !rb,#%11111111                      ;Port B bits 0-7 to input
MODE $09                                ;Set Mode for Pending register
MOV !rb,%00000000                       ;Clear register to allow new interrupts

Main
NOP                                     ;rest of main routine goes here
JMP Main
```

SX Special Features and Coding Tips

The following are points to remember with Wake-Up Interrupts:

- The interrupt routine must be located starting at address \$0 in the SX program.
- It is up to the SX program to clear the bits of the Pending register when future interrupts on that pin are desired. This should normally be done as part of the interrupt routine. The `MOV !rb, #%00000000` instruction effectively clears all bits of the Pending register at the same time that it stores the current edge detection status in W.
- The SX chip will activate the interrupt routine exactly 5 clock cycles in Turbo mode or exactly 10 clock cycles in Compatible mode after a Wake-Up Edge Detection event occurs. This deterministic feature allows for nearly jitter-free interrupt response. Latency may vary by as much as +1 instruction cycle when interrupting on external asynchronous events, thus a high clock speed may be necessary to lessen the effects.
- If multiple interrupt pins are required, the SX chip may not be able to properly process them in certain situations. See Interrupts, below, for more information.
- An edge-detection interrupt event will wake up the SX chip from a SLEEP mode.

See Interrupts, below, for more information.

I/O pins 0 through 2 in port B can be set for comparator operation. This can be configured by writing to the EN and OE bits of the Comparator register (CMP_B) and monitored by reading the RES bit. The comparator mode is disabled by default. Comparator mode can be activated for all three pins, regardless of pin direction, but really matters only when pin 1 and 2 are set to input mode (pin 0 can optionally be set to output the comparative result). By configuring Comparator mode, the SX chip can quickly determine logical differences between two signals and even indicate those differences for external circuitry.

When Comparator mode is activated, the RES bit in the Comparator register indicates the result of the compare. A high bit (1) indicates the

Comparator

7: SX Special Features and Coding Tips

voltage on pin 2 is higher than that of pin 1, a low bit (0) indicates the voltage on pin 2 is lower than that of pin 1. If the OE bit (Output Enable) of the Comparator register is cleared, output pin 0 of port B reflects the state of the RES bit.

To configure port B I/O pins 0 through 2 for Comparator mode:

- 1) Set the MODE register to \$08 (the value for Comparator register configuration).
- 2) Use the port configuration instruction to enable the Comparator and, optionally, the result output on pin 0.
- 3) Set I/O pin directions appropriately.

The following code snippet demonstrates this:

(See Help_Code\Comparator.src on the diskette for full example)

```

;Comparator Configuration
MODE  $08          ;Set Mode to Comparator configuration
MOV   !rb,#%00000000 ;Enable comparator and result output
MODE  $0F          ;Set Mode to allow Direction configuration
MOV   !rb,#%11111110 ;Port B bits 1-7 to input, bit 0 to input

Main
MODE  $08
MOV   !rb,#$00
JMP   Main          ;Here, bit 0 of W holds result of compare
```

The following are points to remember with Comparator mode:

- Port B I/O pins 1 and 2 are the comparator inputs and I/O pin 0 is, optionally, the comparator result output.
- Port B I/O pin 0 may be used as a normal I/O pin by setting the OE bit of the Comparator register.
- The comparator is independent of the clock source and thus will operate even if the SX chip is halted or in SLEEP mode. To avoid spurious current draw during SLEEP mode, disable the comparator.

SX Special Features and Coding Tips

All About Interrupts

The SX chip allows for up to nine sources of interrupts; eight external and one internal. Any or all of the port B I/O pins can be configured as external interrupts. See Wake-Up (Interrupt) on Edge Detection, above, for information on configuring external interrupts. The internal interrupt can be configured to occur upon a rollover condition within the Real Time Clock Counter (RTCC) register. A special return-from-interrupt command may also be used to adjust the value of the RTCC to cause interrupts to occur at a specific time interval. See RTCC Rollover Interrupts, below, for more information.

These interrupt options can be very powerful features but can also cause havoc if not configured or understood properly. If using interrupts of any kind is desired, the following items should be reviewed.

- **Interrupt Vector:** The interrupt vector in the SX chip points to address \$0 and is not configurable. The interrupt routine must reside at location \$0 to be properly executed upon an interrupt event.
- **Auto Interrupt Disable:** As soon as an interrupt occurs, additional interrupts are automatically ignored by the SX chip until the interrupt routine is completed. This prevents the interrupt routine from being interrupted and prevents the loss of return vector data. This is also one of the most important considerations when working with interrupts; you can not immediately (without jitter) process more than one interrupt at a time. **Note: Should additional interrupts occur, the SX chip does not automatically queue up interrupts for future processing. See Interrupt Queuing, below, for more information.**
- **Latency Delays:** When an interrupt occurs, there is a latency delay before the interrupt routine is actually activated. For the internal RTCC rollover, this latency is exactly 3 clock cycles in Turbo mode and 8 clock cycles in Compatible mode. For the external interrupts, the latency delay is exactly 5 clock cycles in Turbo mode and 10 clock cycles in Compatible mode. Latency may vary by as much as +1 instruction cycle when interrupting on external asynchronous

7: SX Special Features and Coding Tips

events, thus a high clock speed may be necessary to lessen the effects.

- **Interrupt Routine Size:** Normally it is a requirement for an application to process every interrupt without missing any. To ensure this happens, the longest path through the interrupt routine must take less time than the shortest possible delay between interrupts.
- **Interrupt Queuing:** If an external interrupt occurs during the interrupt routine, the pending register will be updated but the trigger will be ignored unless interrupts had first been turned off at the beginning of the routine and turned on again at the end. This also requires that the new interrupt doesn't occur before interrupts are turned off in the interrupt routine. If there is a possibility of extra interrupts occurring before they can be disabled, the SX will miss those interrupt triggers.
- **Multiple Interrupts:** Using more than one interrupt, such as multiple external interrupts or both RTCC and external interrupts, can result in missed or, at best, jittery interrupt handling should one occur during the processing of another.
- **Clearing Pending Bits:** When handling external interrupts, the interrupt routine should clear at least one pending register bit. The bit that is cleared should represent the interrupt being handled in order for the next interrupt to trigger.
- **Debugging Interrupts:** The SX chip may act strangely while debugging code that contains interrupts. The SX chip may or may not enter the RTCC interrupt routine (and will never enter a MIWU interrupt) while using the Hop, Jog, Step or Walk functions. This is due to the SX chip giving higher priority to the SX-Key than its internal interrupt flags. **If interrupt code needs to be debugged or verified, place a BREAK directive, or a breakpoint, in an appropriate place within the interrupt routine and use the Run or Poll functions.**

SX Special Features and Coding Tips

The SX chip can be set to cause an interrupt upon rollover of the Real Time Clock Counter (RTCC). By configuring an interrupt on RTCC rollover, the SX chip can perform an operation at a predefined time interval in a deterministic fashion. This can be configured by setting the OPTIONX fuse (in the DEVICE directive) and writing to the RTI, RTS and RTE bits of the Option register (OPTION). The RTCC rollover interrupt is disabled by default.

RTCC Rollover Interrupts

To configure the RTCC rollover interrupt:

- 1) Set the OPTIONX fuse in the DEVICE line.
- 2) Write to the RTI, RTS and RTE bits of the OPTION register to enable RTCC interrupts. For RTI, a high bit (1) disables RTCC rollover interrupts and a low bit (0) enables RTCC rollover interrupts. For RTS, a high bit (1) selects incrementing RTCC on internal clock cycle and a low bit (0) increments RTCC on the RTCC pin transitions. For RTE, a high bit (1) selects incrementing on low-to-high transition and a low bit (0) increments on a high-to-low transition.

The following code snippet demonstrates this:

(See Help_Code\RTCC_Interrupt.src on the diskette for full example)

```
DEVICE OPTIONX
RESET Start

Interrupt                                ;Interrupt routine (must be at address $0)
    ORG    $0
    RETI                                ;rest of interrupt routine goes here

    ;RTCC Rollover Interrupt Configuration
Start
    MOV    !OPTION,%%10011111          ;Enable RTCC rollover interrupt
                                           ;RTCC inc on clock, no prescale
Main
    NOP
    JMP    Main                        ;rest of main routine goes here
```

The above code will cause the interrupt routine to be executed once every 256 clock cycles (when RTCC rolls over from 255 to 0). A different return-from-interrupt command called RETIW can be used, however, to customize the time interval (cycle interval) in which the

7: SX Special Features and Coding Tips

interrupt executes. RETIW, like RETI, causes a return from the interrupt routine. RETIW has the additional effect of adding the contents of W to the RTCC register upon return. By moving a negative number into W just before executing an RETIW, the RTCC will be backed-off by the designated number of cycles. This method also has the benefit of compensating for the number of cycles spent in the interrupt routine.

For example, if the interrupt routine should be executed once every 50 cycles, use the following two lines of code in place of the RETI command in the listing above:

```
MOV    W,#-50
RETIW
```

Of course, for this to work properly the interrupt routine must take 46 cycles or less (see below for cycle bandwidth calculation). Even if the interrupt routine contained multiple paths of execution, due to compare-jump instructions, and each path consumed a different number of clock cycles, the interrupt would still execute once every 50 cycles. Table 7.3 demonstrates the effects on the RTCC if the interrupt routine contained two possible paths of execution (path 1 is 28 cycles and path 2 is 15 cycles):

Table 7.3: Example RTCC values during interrupts.

Event	Path 1 (28 cycles)	Path 2 (15 cycles)
1) RTCC rolls over	RTCC = 0	RTCC = 0
2) 3 cycles required to enter interrupt routine	RTCC = 3	RTCC = 3
3) Interrupt routine executes	RTCC = 31	RTCC = 18
4) -50 (206 in twos-compliment) is added to RTCC	RTCC = 237	RTCC = 224
5) RTCC rolls over again in exactly 19 additional cycles (Path 1) or 32 additional cycles (Path 2)	Total Cycles = 31 + 19 = 50	Total Cycles = 18 + 32 = 50

By adjusting the value in W before the RETIW command, various amounts of cycle bandwidth will be allocated to the main routines. Normally this won't be a problem but care should be taken to ensure that the RTCC doesn't rollover too often, causing little or no cycle time to be allocated to the main routines. If the RTCC adjustment value is too small for the size of the interrupt routine, the main routine may eventually hang up, may not execute at all, or the interrupt routine will

SX Special Features and Coding Tips

miss the rollover and only execute every 256 cycles. Use the following equation as a general rule-of-thumb when determining the minimum adjustment value:

Minimum RTCC Adjustment Value = $-(\text{max cycles for interrupt} + 6)$

The 6 in this equation accounts for the number of cycles required to enter the interrupt routine (3 cycles) plus the number of additional cycles needed to complete the longest command (3 cycles extra to finish an IREAD). If the IREAD command is not used in the main program, a value of $-(\text{maximum cycles for interrupt} + 4)$ is the minimum, allowing for only a single instruction in the main routine to be executed between interrupts.

As an example, the following interrupt routine takes 4 cycles to execute:

```
Interrupt
    MOV    W,#-8        ;1 cycle
    RETIW                ;3 cycles
```

The adjustment value of -8 , which is $-(\text{max cycles for interrupt} + 4)$, will cause the interrupt routine to execute every 8 cycles and will only allow one single-cycle or one three-cycle instruction in the main routine to execute between interrupts. If the main routine contained an IREAD command, however, the main routine would execute one instruction between interrupts until it reached the IREAD, at which point it would get eternally stuck, and only the interrupt would continue. As another example, if the adjustment value was -7 , this would be too small an adjustment and would cause the interrupt routine to execute, but no instructions in the main routine would execute at all.

The following are points to remember with Wake-Up Interrupts:

- The interrupt routine must be located starting at address \$0 in the SX program.
- The interrupt routine should take a maximum of 6 cycles less than the desired cycle time slot. (ie: if the interrupt should execute once every 20 cycles, it needs to be less than 15 cycles in size).

7: SX Special Features and Coding Tips

- The SX chip will activate the interrupt routine exactly 3 clock cycles (Turbo) or 8 clock cycles (Compatible) after an RTCC rollover event occurs. This deterministic feature allows for jitter-free interrupt response.
- An RTCC rollover interrupt event will not occur during SLEEP mode and thus can not wake up the SX chip from a SLEEP mode.

Data Tables

Creating Tables

Tables of 8-bit or 12-bit data may be stored in the unused program space of the SX chip. Tables of data may be necessary in cases where a set of data can not be calculated by an equation, or will take too long to calculate. There are two methods available to store data tables in the SX chip.

If only 8-bit data is required, the RETW method may be used to create the data table. This method uses a set of RETW commands which each hold an 8-bit data value as their operand. The table is preceded by a JMP PC+W command. By moving an index value to W and then CALLing the first line of the table, which is the JMP command, W is added to the program counter and upon the next clock cycle, the proper RETW command is executed. RETW simply moves its operand to W and then returns to the line after the CALL.

To create an 8-bit data table with the RETW command:

- 1) Set the table's location, and insert a label and a JMP PC+W command at the start of the table.
- 2) Add as many RETW commands as necessary.
- 3) When data is needed from the table, move the index value of the desired item to W and CALL the table. Upon returning, the 8-bit value is in W.

SX Special Features and Coding Tips

The following code snippet demonstrates this:

(See Help_Code\RETW_Table.src on the diskette for full example)

```
RESET Initialize

Idx EQU $08 ;Define index symbol

;8-bit data table
Table ORG $0
      JMP PC+W ;Jump into the table
      RETW 'ABCDEFGH' ;Store text
      RETW 10,100,255,0 ;Store numbers
Initialize
      MOV Idx,$FF ;Reset table index

Main INC Idx ;Increment table index
      MOV W,Idx
      CALL Table ;Retrieve data
```

If 8-bit or 12-bit data is required, the DW method may be used to create the data table. This method uses a set of DW (Define Word) directives each of which place a 12-bit data value in program memory. By moving a 12-bit index value to M and W (upper 4 bits of address in M) and executing an IREAD command, M and W are replaced with the 12-bit data value.

To create an 8-bit or 12-bit data table with the DW directive:

- 1) Set the table's location and insert a label.
- 2) Add as many DW directives as necessary.
- 3) When data is needed from the table, move the index value of the desired item (in relation to the label) to M and W and execute an IREAD. Upon returning, the 12-bit value is in M and W.

7: SX Special Features and Coding Tips

The following code snippet demonstrates this:
(See Help_Code\IREAD_Table.src on the diskette for full example)

```
RESET Initialize

      Idx    EQU    $08    ;Define index symbol
      ;8-bit data table
      ORG    $0
Table DW    'ABCDEFGH'    ;Store text
      DW    10,300,4095,0 ;Store numbers

Initialize
      MOV    Idx,#Table    ;Reset table index

Main   MOV    M,#Table>>8    ;upper 4-bits of table address
      MOV    W,Idx          ;lower 8-bits of table index
      IREAD                ;Retrieve data
      {use the data}
      INC    Idx            ;Increment table index
      CJNE   Idx,#11,Main    ;Continue
```

Both table methods shown above will only access a maximum of 256 elements, however, the DW method can easily be modified to access every possible address. If speed is desired, the DW method, above, is 4 cycles shorter per element than the RETW method. With clever modifications, both methods could be used for more than 8 or 12-bit.

Branching Across Pages

Dealing with Code Pages

The SX chip's program memory is organized into pages of 512 words each. If a program won't fit within a single page, special care must be taken when branching across page boundaries to avoid misdirected jumps.

All instructions that perform a jump, except JMP W, JMP PC+W and LJMP, use a 9-bit address as the operand. This limits the jump range to the current page only (512 words). To jump across a page boundary, the page select bits (the upper bits of the Status register) must first be set to the appropriate page before executing the jump. The SX assembler provides a convenient method of doing this, as shown below.

SX Special Features and Coding Tips

To jump across page boundaries:

- 1) Specify the jump command (JMP, JB, CJE, etc) with the page-set option (the @ sign preceding the address).

The following code snippet demonstrates this:

(See Help_Code\Page_Jumping.src on the diskette for full example)

```
Start   ORG   $0           ;This routine is in page 0
        JMP   @Routine     ;Jump to proper page
        JMP   Start

Routine ORG   $200         ;This routine is in page 1
        JMP   @Start       ;Jump back to proper page
```

The @ symbol preceding the address causes the SX editor to insert a PAGE instruction just before the JMP to set the page select bits appropriately. The second JMP, in line three, does not require an @ symbol since the destination address is within the current page. If the @ was left out of line two, the SX would jump to address \$000 instead of \$200. See Jumping Across Pages in Appendix F for more information.

Calling subroutines can pose even more boundary problems than jumping across pages. The CALL instruction uses only an 8-bit address as the operand (the 9th bit of the address is always cleared). This limits the calling destination to the first 256 words of the current page.

Calling Across Pages with Jump Tables

Because it is sometimes impossible to organize all subroutines within such a tight space, a common practice is to make use of a subroutine jump table. The jump table consists of a list of JMP commands to various subroutines and is located within the first 256 words of the page. The CALL instructions can simply call the proper location within the jump table and code execution jumps to the appropriate subroutine, even if it exists in different pages or above the 256 word barrier.

7: SX Special Features and Coding Tips

To call subroutines across page boundaries:

- 1) Design a jump table with the page-set option (the @ sign preceding the addresses).
- 2) Place the subroutines in any desired location being sure to end them with RETP.
- 3) Call the subroutine's alias-name in the jump table.

The following code snippet demonstrates this:

(See Help_Code\Page_Calling.src on the diskette for full example)

```

                ORG    $0                ;This routine is in page 0
                ;Define the subroutine jump-table
Sub1    JMP    @_Sub1                ;Set page and jump
Sub2    JMP    @_Sub2

                ;Start of main routines
Start   CALL    @Sub1                ;Call the Jump Table
        JMP    @Continue

                ORG    $200              ;This routine is in page 1
Continue CALL    @Sub2                ;Call the Jump Table
        JMP    @Start

                ORG    $400              ;This routine is in page 2
_Sub1   ;Subroutine 1 code goes here
        RETP                    ;Return and reset page

_Sub2   ;Subroutine 2 code goes here
        RETP                    ;Return and reset page
```

The first CALL in the Start routine calls the *Sub1* address in the jump table. The JMP command at *Sub1* then jumps to the *_Sub1* subroutine (in page 2) which eventually returns to the line following the CALL. The RETP command used to return from the subroutine resets the page select bits to the page of the calling routine (exactly as intended).

The @ symbol preceding the addresses causes the SX editor to insert a PAGE instruction just before the JMP and CALL commands to set the page select bits appropriately. The first CALL, in the Start routine, would function the same without an @ symbol, as shown above, since

SX Special Features and Coding Tips

the destination address is within the current page. See Calling Across Pages in Appendix F for more information.

Appendix A: SX Features

Introduction

The SX chip is a fully static CMOS MPU conservatively rated for DC to 50 MHz operation. The SX provides 2K words of on-chip EE/Flash program memory. In Turbo mode, all instructions are single cycle except program branches, which take three cycles, and IREAD, which takes four cycles.

CPU Features

- Single cycle instruction execution (20 ns cycle time @ 50 MHz)
- DC to 50 MHz operation
- User selectable clock options: (Internal R/C, External R/C, resonator, or crystal oscillator)
- Internal R/C oscillator ($4 \text{ MHz} / 2^{0-7}$, $\pm 8\%$ accuracy)
- 43 instructions (designed for compatibility with the PIC16C5x series; 10 new instructions for improved code efficiency)
- 2048 x 12-bits EE/Flash program memory rated for 10,000 rewrite cycles
- Up to 137 bytes of directly, or indirectly, addressable RAM
- Selectable 8-level hardware stack
- Fixed interrupt response time: 60 ns internal, 100 ns external
- Hardware context save/restore of PC, W, STATUS, and FSR on interrupt.
- Multi-Input Wake-Up (MIWU) on 8 pins
- In-system programming via OSC pins
- Single-step and breakpoint debugging via OSC2 pin
- Analog comparator (RB0 out, RB1 in-, RB2 in+)
- Built-in brown-out detector (On/Off, 4.2V)
- W mappable into RTCC space for flexibility
- 1998 UL compliance and fast lookup provided through run-time readable code

Peripheral and I/O Features

- Every pin programmable as input or output
- Inputs are each TTL or CMOS level selectable
- All pins include selectable internal pull-ups ($\sim 20 \text{ k}\Omega$ to V_{DD})

SX Features

- RB, RC inputs each selectable as Schmitt Trigger
- All outputs capable of sinking / sourcing 30 mA
- Symmetrical drive on RA outputs (same Vdrop +/-)

Appendix B: SX Instruction Set

Parallax Instruction Set Summary

This chart lists the general syntax for the entire Parallax SX instruction set.
{ } = optional, [] = choice of one of many.

ADD	dest, src	LCALL	addr11
ADDB	dest, {/} src_bit	LJMP	addr11
AND	dest, src	LSET	addr11
BANK	dest	MODE	lit
CALL	addr8	MOV	{!}dest, {!/,--,++,<<,>>,<>} src {-dest}
CJA	op1, op2, addr	MOVB	dest_bit, {/}src_bit
CJAE	op1, op2, addr	MOVSZ	dest, [-- ++]src
CJB	op1, op2, addr	NOP	
CJBE	op1, op2, addr	NOT	dest
CJE	op1, op2, addr	OR	dest, src
CJNE	op1, op2, addr	PAGE	addr12
CLC		RET	
CLR	dest	RETI	
CLRB	dest_bit	RETIW	
CLZ		RETP	
CSA	op1, op2	RETW	lit{,lit...}
CSAE	op1, op2	RL	dest
CSB	op1, op2	RR	dest
CSBE	op1, op2	SB	src_bit
CSE	op1, op2	SC	
CSNE	op1, op2	SETB	dest_bit
DEC	dest	SKIP	
DECSZ	dest	SLEEP	
DJNZ	dest, addr	SNB	src_bit
IJNZ	dest, addr	SNC	
INC	dest	SNZ	
INCSZ	dest	STC	
IREAD		STZ	
JB	op_bit, addr	SUB	dest, src
JC	addr	SUBB	dest, {/} src_bit
JMP	addr	SWAP	dest
JNB	op_bit, addr	SZ	
JNC	addr	TEST	dest
JNZ	addr	XOR	dest, src
JZ	addr		

SX Instruction Set

Single-Word Instructions

All instructions in this list consume only one word of SX EEPROM space.

ADD	fr, W	MOV	!OPTION, W
ADD	W, fr	MOV	!port, W
AND	fr, W	MOVSZ	W, ++fr
AND	W, fr	MOVSZ	W, --fr
AND	W, #literal	NOP	
BANK	fr	NOT	fr
CALL	addr	NOT	W
CLC		OR	fr, W
CLR	fr	OR	W, fr
CLR	W	OR	W, #literal
CLR	!WDT	PAGE	addr
CLRB	op.bit	RET	
CLZ		RETI	
DEC	fr	RETIW	
DECSZ	fr	RETP	
INC	fr	RL	fr
INCSZ	fr	RR	fr
IREAD		SB	op.bit
JMP	addr	SC	
JMP	W	SETB	op.bit
JMP	PC+W	SKIP	
MODE	literal	SLEEP	
MOV	fr, W	SNB	op.bit
MOV	W, fr	SNC	
MOV	W, /fr	SNZ	
MOV	W, fr-W	STC	
MOV	W, ++fr	STZ	
MOV	W, --fr	SUB	fr, W
MOV	W, <<fr	SWAP	fr
MOV	W, >>fr	SZ	
MOV	W, <>fr	TEST	fr
MOV	W, #literal	TEST	W
MOV	W, M	XOR	fr, W
MOV	M, #literal	XOR	W, fr
MOV	M, W	XOR	W, #literal

Appendix B: SX Instruction Set

Multi-Word Instructions

All instructions in this list may consume more than one word of SX EEPROM space.

ADD	fr, #literal	DJNZ	fr, addr
ADD	fr1, fr2	IJNZ	fr, addr
ADDB	fr, op.bit	JB	op.bit, addr
ADDB	fr, /op.bit	JC	addr
AND	fr, #literal	JNB	op.bit, addr
AND	fr1, fr2	JNC	addr
CJA	fr, #literal, addr	JNZ	addr
CJA	fr1, fr2, addr	JZ	addr
CJAE	fr, #literal, addr	LCALL**	addr
CJAE	fr1, fr2, addr	LJMP**	addr
CJB	fr, #literal, addr	LSET*	addr
CJB	fr1, fr2, addr	MOV	fr, #literal
CJBE	fr, #literal, addr	MOV	fr1, fr2
CJBE	fr1, fr2, addr	MOV	fr, M
CJE	fr, #literal, addr	MOV	M, fr
CJE	fr1, fr2, addr	MOV	!OPTION, fr
CJNE	fr, #literal, addr	MOV	!OPTION, #literal
CJNE	fr1, fr2, addr	MOV	!port, fr
CSA	fr, #literal	MOV	!port, #literal
CSA	fr1, fr2	MOVB	op.bit1, op.bit2
CSAE	fr, #literal	MOVB	op.bit1, /op.bit2
CSAE	fr1, fr2	OR	fr, #literal
CSB	fr, #literal	OR	fr1, fr2
CSB	fr1, fr2	RETW***	
CSBE	fr, #literal	SUB	fr1, #literal
CSBE	fr1, fr2	SUB	fr1, fr2
CSE	fr, #literal	SUBB	fr, op.bit
CSE	fr1, fr2	SUBB	fr, /op.bit
CSNE	fr, #literal	XOR	fr, #literal
CSNE	fr1, fr2	XOR	fr1, fr2

* zero to three words. ** one to four words. *** one or more words.

SX Instruction Set

Instruction Set Quick Reference

This chart is a quick reference to command syntax, size, cycle time and CARRYX sensitivity.

{ } = optional, [] = choice of one of many, () = non-Turbo cycles, #, # = no branch / branch cycles.

Command	Words	Cycles	Command	Words	Cycles
ADD fr, W	1	1(4)	MOV fr, [#literal fr2 M]	2	2(8)
ADD fr, [#literal fr2]	2	2(8)	MOV W, {/ ++ -- << >> <>}fr	1	1(4)
ADD W, fr	1	1(4)	MOV* W, fr-W	1	1(4)
ADDB fr, {/} op.bit	2	2(8)	MOV W, [#literal M]	1	1(4)
AND fr, W	1	1(4)	MOV M, fr	2	2(8)
AND fr, [#literal fr2]	2	2(8)	MOV M, [#literal W]	1	1(4)
AND W, [#literal fr]	1	1(4)	MOV !OPTION, [fr #literal]	2	2(8)
BANK fr	1	1(4)	MOV !OPTION, W	1	1(4)
CALL addr	1	3(8)	MOV !port, [fr #literal]	2	2(8)
CLC	1	1(4)	MOV !port, W	1	1(4)
CLR [fr W !WDT]	1	1(4)	MOVB op.bit1, {/}op.bit2	4	4(16)
CLRB op.bit	1	1(4)	MOVSZ W, [++ --]fr	1	1,2(4,8)
CLZ	1	1(4)	NOP	1	1(4)
CJA* fr, [#literal fr2], addr	4	4,6(16,20)	NOT [fr W]	1	1(4)
CJAE* fr, [#literal fr2], addr	4	4,6(16,20)	OR fr, W	1	1(4)
CJB* fr, [#literal fr2], addr	4	4,6(16,20)	OR fr, [#literal fr2]	2	2(8)
CJBE* fr, [#literal fr2], addr	4	4,6(16,20)	OR W, [fr #literal]	1	1(4)
CJE* fr, [#literal fr2], addr	4	4,6(16,20)	PAGE addr	1	1(4)
CJNE* fr, [#literal fr2], addr	4	4,6(16,20)	RET	1	3(8)
CSA* fr, [#literal fr2]	3	3,4(12,16)	RETI	1	3(8)
CSAE* fr, [#literal fr2]	3	3,4(12,16)	RETIW	1	3(8)
CSB* fr, [#literal fr2]	3	3,4(12,16)	RETP	1	3(8)
CSBE* fr, [#literal fr2]	3	3,4(12,16)	RETW	1+	3(8)
CSE* fr, [#literal fr2]	3	3,4(12,16)	RL fr	1	1(4)
CSNE* fr, [#literal fr2]	3	3,4(12,16)	RR fr	1	1(4)
DEC fr	1	1(4)	SB op.bit	1	1,2(4,8)
DECSZ fr	1	1,2(4,8)	SC	1	1,2(4,8)
DJNZ fr, addr	2	2,4(8,12)	SETB op.bit	1	1(4)
INC fr	1	1(4)	SKIP	1	2(8)
INCSZ fr	1	1,2(4,8)	SLEEP	1	1(4)
IJNZ fr, addr	2	2,4(8,12)	SNB op.bit	1	1,2(4,8)
IREAD	1	4(16)	SNC	1	1,2(4,8)
JB op.bit, addr	2	2,4(8,12)	SNZ	1	1,2(4,8)
JC addr	2	2,4(8,12)	STC	1	1(4)
JMP [addr W]	1	3(8)	STZ	1	1(4)
JMP* PC+W	1	3(8)	SUB* fr1, [#literal fr2]	2	2(8)
JNB op.bit, addr	2	2,4(8,12)	SUB* fr, W	1	1(4)
JNC addr	2	2,4(8,12)	SUBB fr, {/}op.bit	2	2(8)
JNZ addr	2	2,4(8,12)	SWAP fr	1	1(4)
JZ addr	2	2,4(8,12)	SZ	1	1,2(4,8)
LCALL addr	1-4	3-6(8-20)	TEST [fr W]	1	1(4)
LJMP addr	1-4	3-6(8-20)	XOR fr, [#literal fr2]	2	2(8)
LSET addr	0-3	0-3(0-12)	XOR fr, W	1	1(4)
MODE literal	1	1(4)	XOR W, [fr #literal]	1	1(4)
MOV fr, W	1	1(4)			

* adversely affected by carry flag when CARRYX specified. See instruction syntax for more information.

Appendix B: SX Instruction Set

The columns of each instruction definition table in this appendix contain important information about the instructions behavior, size and structure.

The Command column lists all the available forms of the given command. The operands in lower-case letters indicate a symbol or value should be inserted in their place. The operands in upper-case letters should be entered exactly as seen. For example, the following form of the MOV command:

MOV !OPTION, #literal

should be entered into code (assuming \$A5 is the desired literal) as follows:

MOV !OPTION, #\$A5

The operands are described in Table B.1, below.

Table B.1: Symbols in the command column.

Symbol	Definition
addr	An address symbol or value
fr	A file register
#literal	A literal value ('#' must precede value)
M	The mode register
op.bit	The specified bit of the specified operand
!OPTION	The option register
PC	The program counter
!port	The specified I/O port data direction reg.
W	The working register
!WDT	The watchdog register

The Words column indicates the number of 12-bit EEPROM words consumed by the instruction.

The Cycles column indicates the number of clock cycles the given instruction will take. The number outside of parenthesis is the number of cycles in Turbo mode. The number inside parenthesis is the number of cycles in non-Turbo mode (compatibility mode).

The Affects column indicates the flags and registers the given instruction affects. These flags and registers are described in Table B.2, below.

SX Instruction Set

Symbol	Definition
C	The carry flag
DC	The digit-carry flag
FSR	The file select register
fr	The file register
M	The mode register
op.bit	The specified bit of the specified operand
OPT	The options register
PC	The program counter
PD	The power-down flag
!port	The specified I/O port data direction reg.
TO	The time-out flag
W	The working register
Z	The zero flag

Table B.2: Symbols in the affects column.

The Coding column indicates how the given command will assemble, including binary values and mnemonic instructions. Some commands assemble into multiple, simpler commands. Table B.3 describes the binary symbols.

Symbol	Definition
b	Bit address
f	File register address
K	Constant

Table B.3: Symbols in the coding column.

Any instruction that performs a skip will only skip one instruction word. To avoid strange results, care must be taken to make sure that a single word instruction immediately follows the skipping instruction.

Many instructions are adversely affected by the carry flag when CARRYX (Add/Sub with C) is specified. Be careful to follow the special notes within the instruction descriptions concerning this.

Appendix B: SX Instruction Set

ADD dest, src

Add src into dest

Command	Words	Cycles	Affects	Coding
1) ADD fr, W	1	1 (4)	fr,C,DC,Z	0001 111f ffff ADD fr,W
2) ADD fr, #literal	2	2 (8)	fr,W,C,DC,Z	1100 kkkk kkkk MOV W,#lit 0001 111f ffff ADD fr,W
3) ADD fr1, fr2	2	2 (8)	fr,W,C,DC,Z	0010 000f ffff MOV W,fr2 0001 111f ffff ADD fr1,W
4) ADD W, fr	1	1 (8)	W,C,DC,Z	0001 110f ffff ADD W,fr

Operation: Src is added into dest. C will be set if an overflow occurs; otherwise, C will be cleared. DC will be set if an overflow occurs in the lower nibble; otherwise, DC will be cleared. Z will be set if the result is 0; otherwise, Z will be cleared. W is left holding the source value in command #2 and #3. **If CARRYX is specified, c is added to result. Insert a CLC before the first Add on a register to avoid strange results.**

ADDB dest, src_bit

Add src_bit into dest

Command	Words	Cycles	Affects	Coding
1) ADDB fr, op.bit	2	2 (8)	fr,Z	0110 bbbf ffff SNB op.bit 0010 101f ffff INC fr
2) ADDB fr, /op.bit	2	2 (8)	fr,Z	0111 bbbf ffff SB op.bit 0010 101f ffff INC fr

Operation: Src_bit is added into dest. If fr is incremented, Z will be set if the result is 0; otherwise, Z will be cleared. This instruction is useful for adding the carry into the upper byte of a double-byte sum after the lower byte has been computed.

AND dest, src

AND src into dest

Command	Words	Cycles	Affects	Coding
1) AND fr, W	1	1 (4)	fr,Z	0001 011f ffff AND fr,W
2) AND fr, #literal	2	2 (8)	fr,W,Z	1100 kkkk kkkk MOV W,#lit 0001 011f ffff AND fr,W
3) AND fr1, fr2	2	2 (8)	fr1,W,Z	0010 000f ffff MOV W,fr2 0001 011f ffff AND fr1,W
4) AND W, fr	1	1 (4)	W,Z	0001 010f ffff AND W,fr
5) AND W, #literal	1	1 (4)	W,Z	1110 kkkk kkkk AND W,#lit

Operation: Src is AND'd into dest. Z will be set if the result is 0; otherwise, Z will be cleared. W is left holding the source value in command #2 and #3.

SX Instruction Set

BANK dest

Set bank select bits

Command	Words	Cycles	Affects	Coding
1) BANK fr	1	1 (4)	FSR	0000 0001 1fff BANK fr

Operation: writes file registers bits 7 through 5 into file select register bits 7 through 5 in preparation for a RAM access across a bank boundary.

CALL addr8

Call subroutine with 8-bit address

Command	Words	Cycles	Affects	Coding
1) CALL addr8	1	3 (8)	PC	1001 kkkk kkkk CALL addr8

Operation: The next instruction address is pushed onto the stack and addr8 is moved to the program counter. The ninth bit of the program counter will be cleared to 0. Therefore, calls are only allowed to the first half of any 512-word page, although the CALL instruction can be anywhere.

CJA op1, op2, addr

Compare op1 to op2 and jump if above

Command	Words	Cycles	Affects	Coding
1) CJA fr, #literal, addr	4	4 or 6 (jump) (16 or 20)	W,C,DC,Z	1100 kkkk kkkk MOV w, #lit^\$FF 0001 110f ffff ADD w, fr 0110 0000 0011 SNC 101k kkkk kkkk JMP addr
2) CJA fr1, fr2, addr	4	4 or 6 (jump) (16 or 20)	W,C,DC,Z	0010 000f ffff MOV w, fr1 0000 100f ffff MOV w, fr2-w 0111 0000 0011 SC 101k kkkk kkkk JMP addr

Operation: op1 is compared to op2. If op1 is greater than op2, a jump to addr is executed. W is left holding the result of $op1 + \sim op2$ in command #1 and $op2 - op1$ in command #2. **If CARRYX is specified, c affects result. Insert a CLC before command #1 and an STC before command #2 to avoid strange results.**

Appendix B: SX Instruction Set

CJAE op1, op2, addr Compare op1 to op2 and jump if above or equal

Command	Words	Cycles	Affects	Coding
1) CJAE fr, #literal, addr	4	4 or 6 (jump) (16 or 20)	W,C,DC,Z	1100 kkkk kkkk MOV w,#lit 0000 100f ffff MOV w,fr-w 0110 0000 0011 SNC 101k kkkk kkkk JMP addr
2) CJAE fr1, fr2, addr	4	4 or 6 (jump) (16 or 20)	W,C,DC,Z	0010 000f ffff MOV w,fr2 0000 100f ffff MOV w,fr1-w 0110 0000 0011 SNC 101k kkkk kkkk JMP addr

Operation: op1 is compared to op2. If op1 is greater than or equal to op2, a jump to addr is executed. W is left holding the result of op2 - op1. **If CARRYX is specified, c affects result. Insert an STC before command to avoid strange results.**

CJB op1, op2, addr Compare op1 to op2 and jump if below

Command	Words	Cycles	Affects	Coding
1) CJB fr, #literal, addr	4	4 or 6 (jump) (16 or 20)	W,C,DC,Z	1100 kkkk kkkk MOV w,#lit 0000 100f ffff MOV w,fr-w 0111 0000 0011 SC 101k kkkk kkkk JMP addr
2) CJB fr1, fr2, addr	4	4 or 6 (jump) (16 or 20)	W,C,DC,Z	0010 000f ffff MOV w,fr2 0000 100f ffff MOV w,fr1-w 0111 0000 0011 SC 101k kkkk kkkk JMP addr

Operation: op1 is compared to op2. If op1 is less than op2, a jump to addr is executed. W is left holding the result of op2 - op1. **If CARRYX is specified, c affects result. Insert an STC before command to avoid strange results.**

SX Instruction Set

CJBE op1, op2, addr Compare op1 to op2 and jump if below or equal

Command	Words	Cycles	Affects	Coding
1) CJBE fr, #literal, addr	4	4 or 6 (jump) (16 or 20)	W,C,DC,Z	1100 kkkk kkkk MOV w, #lit^\$FF 0001 110f ffff ADD w, fr 0111 0000 0011 SC 101k kkkk kkkk JMP addr
2) CJBE fr1, fr2, addr	4	4 or 6 (jump) (16 or 20)	W,C,DC,Z	0011 000f ffff MOV w, fr1 0000 100f ffff MOV w, fr2-w 0110 0000 0011 SNC 101k kkkk kkkk JMP addr

Operation: op1 is compared to op2. If op1 is less than or equal to op2, a jump to addr is executed. W is left holding the result of $\sim op2 + op1$ in command #1 and $op2 - op1$ in command #2. **If CARRYX is specified, c affects result. Insert a CLC before command #1 and an STC before command #2 to avoid strange results.**

CJE op1, op2, addr Compare op1 to op2 and jump if equal

Command	Words	Cycles	Affects	Coding
1) CJE fr, #literal, addr	4	4 or 6 (jump) (16 or 20)	W,C,DC,Z	1100 kkkk kkkk MOV w, #lit 0000 100f ffff MOV w, fr-w 0110 0100 0011 SNZ 101k kkkk kkkk JMP addr
2) CJE fr1, fr2, addr	4	4 or 6 (jump) (16 or 20)	W,C,DC,Z	0010 000f ffff MOV w, fr2 0000 100f ffff MOV w, fr1-w 0110 0100 0011 SNZ 101k kkkk kkkk JMP addr

Operation: op1 is compared to op2. If op1 is equal to op2, a jump to addr is executed. W is left holding the result of $op1 - op2$. **If CARRYX is specified, c affects result. Insert an STC before command to avoid strange results.**

Appendix B: SX Instruction Set

CJNE op1, op2, addr Compare op1 to op2 and jump if not equal

Command	Words	Cycles	Affects	Coding
1) CJNE fr, #literal, addr	4	4 or 6 (jump) (16 or 20)	W,C,DC,Z	1100 kkkk kkkk MOV W,#lit 0000 100f ffff MOV W,fr-w 0111 0100 0011 SZ 101k kkkk kkkk JMP addr
2) CJNE fr1, fr2, addr	4	4 or 6 (jump) (16 or 20)	W,C,DC,Z	0010 000f ffff MOV W,fr2 0000 100f ffff MOV W,fr1-w 0111 0100 0011 SZ 101k kkkk kkkk JMP addr

Operation: op1 is compared to op2. If op1 is not equal to op2, a jump to addr is executed. W is left holding the result of op1 - op2. **If CARRYX is specified, c affects result. Insert an STC before command to avoid strange results.**

CLC Clear carry

Command	Words	Cycles	Affects	Coding
1) CLC	1	1 (4)	C	0100 0000 0011 CLC

Operation: C is cleared to 0.

CLR dest Clear dest

Command	Words	Cycles	Affects	Coding
1) CLR fr	1	1 (4)	fr,Z	0000 011f ffff CLR fr
2) CLR W	1	1 (4)	W,Z	0000 0100 0000 CLR w
3) CLR !WDT	1	1 (4)	TO,PD	0000 0000 0100 CLR wdt

Operation: dest is cleared to 0. Z is set to 1 in command #1 and #2 while TO and PD are set to 1 in command #3. Prescaler is also cleared in command #3, if assigned.

CLRB dest_bit Clear dest_bit

Command	Words	Cycles	Affects	Coding
1) CLRB op.bit	1	1 (4)	op.bit	0100 bbbf ffff CLRB op.bit

Operation: dest_bit is cleared to 0.

SX Instruction Set

CLZ

Clear zero

Command	Words	Cycles	Affects	Coding
1) CLZ	1	1 (4)	Z	0100 0100 0011 CLZ

Operation: Z is cleared to 0.

CSA op1, op2

Compare op1 to op2 and skip if above

Command	Words	Cycles	Affects	Coding
1) CSA fr, #literal	3	3 or 4 (skip) (12 or 16)	W,C,DC,Z	1100 kkkk kkkk MOV W,#lit^\$FF 0001 110f ffff ADD W,fr 0111 0000 0011 SC
2) CSA fr1, fr2	3	3 or 4 (skip) (12 or 16)	W,C,DC,Z	0010 000f ffff MOV W,fr1 0000 100f ffff MOV W,fr2-W 0110 0000 0011 SNC

Operation: op1 is compared to op2. If op1 is greater than op2, the following instruction word is skipped. W is left holding the result of op1 + ~op2 in command #1 and op2 - op1 in command #2. **If CARRYX is specified, c affects result. Insert a CLC before command #1 and an STC before command #2 to avoid strange results.**

Note: Only one word is skipped by this instruction. **To avoid strange results, make sure that any instruction following CSA is a single-word instruction.**

CSAE op1, op2

Compare op1 to op2 and skip if above or equal

Command	Words	Cycles	Affects	Coding
1) CSAE fr, #literal	3	3 or 4 (skip) (12 or 16)	W,C,DC,Z	1100 kkkk kkkk MOV W,#lit 0000 100f ffff MOV W,fr-W 0111 0000 0011 SC
2) CSAE fr1, fr2	3	3 or 4 (skip) (12 or 16)	W,C,DC,Z	0010 000f ffff MOV W,fr2 0000 100f ffff MOV W,fr1-W 0111 0000 0011 SC

Operation: op1 is compared to op2. If op1 is greater than or equal to op2, the following instruction word is skipped. W is left holding the result of op1 - op2. **If CARRYX is specified, c affects result. Insert an STC before command to avoid strange results.**

Note: Only one word is skipped by this instruction. **To avoid strange results, make sure that any instruction following CSAE is a single-word instruction.**

Appendix B: SX Instruction Set

CSB op1, op2 Compare op1 to op2 and skip if below

Command	Words	Cycles	Affects	Coding
1) CSB fr, #literal	3	3 or 4 (skip) (12 or 16)	W,C,DC,Z	1100 kkkk kkkk MOV W,#lit 0000 100f ffff MOV W,fr-w 0110 0000 0011 SNC
2) CSB fr1, fr2	3	3 or 4 (skip) (12 or 16)	W,C,DC,Z	0010 000f ffff MOV W,fr2 0000 100f ffff MOV W,fr1-w 0110 0000 0011 SNC

Operation: op1 is compared to op2. If op1 is less than op2, the following instruction word is skipped. W is left holding the result of op1 - op2. **If CARRYX is specified, c affects result. Insert an STC before command to avoid strange results.**

Note: Only one word is skipped by this instruction. **To avoid strange results, make sure that any instruction following CSB is a single-word instruction.**

CSBE op1, op2 Compare op1 to op2 and skip if below or equal

Command	Words	Cycles	Affects	Coding
1) CSBE fr, #literal	3	3 or 4 (skip) (12 or 16)	W,C,DC,Z	1100 kkkk kkkk MOV W,#lit^\$FF 0001 110f ffff ADD W,fr 0110 0000 0011 SNC
2) CSBE fr1, fr2	3	3 or 4 (skip) (12 or 16)	W,C,DC,Z	0010 000f ffff MOV W,fr1 0000 100f ffff MOV W,fr2-w 0111 0000 0011 SC

Operation: op1 is compared to op2. If op1 is less than or equal to op2, the following instruction word is skipped. W is left holding the result of op1 + ~op2 in command #1 and op2 - op1 in command #2. **If CARRYX is specified, c affects result. Insert a CLC before command #1 and an STC before command #2 to avoid strange results.**

Note: Only one word is skipped by this instruction. **To avoid strange results, make sure that any instruction following CSBE is a single-word instruction.**

SX Instruction Set

CSE op1, op2 Compare op1 to op2 and skip if equal

Command	Words	Cycles	Affects	Coding
1) CSE fr, #literal	3	3 or 4 (skip) (12 or 16)	W,C,DC,Z	1100 kkkk kkkk MOV W, #lit 0000 100f ffff MOV W, fr-W 0111 0100 0011 SZ
2) CSE fr1, fr2	3	3 or 4 (skip) (12 or 16)	W,C,DC,Z	0010 000f ffff MOV W, fr2 0000 100f ffff MOV W, fr1-W 0111 0100 0011 SZ

Operation: op1 is compared to op2. If op1 is equal to op2, the following instruction word is skipped. W is left holding the result of op1 - op2. **If CARRYX is specified, c affects result. Insert an STC before command to avoid strange results.**

Note: Only one word is skipped by this instruction. **To avoid strange results, make sure that any instruction following CSE is a single-word instruction.**

CSNE op1, op2 Compare op1 to op2 and skip if not equal

Command	Words	Cycles	Affects	Coding
1) CSNE fr, #literal	3	3 or 4 (skip) (12 or 16)	W,C,DC,Z	1100 kkkk kkkk MOV W, #lit 0000 100f ffff MOV W, fr-W 0110 0100 0011 SNZ
2) CSNE fr1, fr2	3	3 or 4 (skip) (12 or 16)	W,C,DC,Z	0010 000f ffff MOV W, fr2 0000 100f ffff MOV W, fr1-W 0110 0100 0011 SNZ

Operation: op1 is compared to op2. If op1 is not equal to op2, the following instruction word is skipped. W is left holding the result of op1 - op2. **If CARRYX is specified, c affects result. Insert an STC before command to avoid strange results.**

Note: Only one word is skipped by this instruction. **To avoid strange results, make sure that any instruction following CSNE is a single-word instruction.**

DEC dest Decrement dest

Command	Words	Cycles	Affects	Coding
1) DEC fr	1	1 (4)	fr,Z	0000 111f ffff DEC fr

Operation: dest is decremented. Z will be set to 1 if the result was 0; otherwise, Z will be cleared to 0. *The MOV W,--fr command is similar to DEC fr, except the result is moved to W.*

Appendix B: SX Instruction Set

DECSZ dest

Decrement dest and skip if zero

Command	Words	Cycles	Affects	Coding
1) DECSZ fr	1	1 or 2 (skip) (4 or 8)	fr	0010 111f ffff DECSZ fr

Operation: dest is decremented. If result is 0, the next instruction word will be skipped. Z will be set to 1 if the result was 0; otherwise, Z will be cleared to 0.

Note: Only one word is skipped by this instruction. **To avoid strange results, make sure that any instruction following DECSZ is a single-word instruction.**

DJNZ dest, addr

Decrement dest and jump if not zero

Command	Words	Cycles	Affects	Coding
1) DJNZ fr, addr	2	2 or 4 (jump) (8 or 12)	fr	0010 111f ffff DECSZ fr 101k kkkk kkkk JMP addr

Operation: dest is decremented. If the result is not 0, a jump to addr is executed.

IJNZ dest, addr

Increment dest and jump if not zero

Command	Words	Cycles	Affects	Coding
1) IJNZ fr, addr	2	2 or 4 (jump) (8 or 12)	fr	0011 111f ffff INC SZ fr 101k kkkk kkkk JMP addr

Operation: dest is incremented. If the result is not 0, a jump to addr is executed.

INC dest

Increment dest

Command	Words	Cycles	Affects	Coding
1) INC fr	1	1 (4)	fr,Z	0010 101f ffff INC fr

Operation: dest is incremented. Z will be set to 1 if the result was 0; otherwise, Z will be cleared to 0. *The MOV W,++fr command is similar to INC fr, except the result is moved to W.*

SX Instruction Set

INCSZ dest

Increment dest and skip if zero

Command	Words	Cycles	Affects	Coding
1) INCSZ fr	1	1 or 2 (skip) (4 or 8)	fr	0011 111f ffff INCSZ fr

Operation: dest is incremented. If result is 0, the next instruction word will be skipped. Z will be set to 1 if the result was 0; otherwise, Z will be cleared to 0.

Note: Only one word is skipped by this instruction. **To avoid strange results, make sure that any instruction following INCSZ is a single-word instruction.**

IREAD

Read instruction at MODE:W into MODE:W

Command	Words	Cycles	Affects	Coding
1) IREAD	1	4 (16)	W	0000 0100 0001 IREAD

Operation: the instruction or value at the 12-bit location MODE:W is read and stored into the MODE:W bits. This instruction can be used to read values in tables created with the DW directive. Use the MODE and MOVE instructions to read and write from the mode register.

JB src_bit, addr

Jump if src_bit is set

Command	Words	Cycles	Affects	Coding
1) JB op.bit, addr	2	2 or 4 (jump) (8 or 12)	none	0110 bbbf ffff SNB op.bit 101k kkkk kkkk JMP addr

Operation: if src_bit is set, a jump to addr is executed.

JC addr

Jump if carry

Command	Words	Cycles	Affects	Coding
1) JC addr	2	2 or 4 (jump) (8 or 12)	none	0110 0000 0011 SNZ 101k kkkk kkkk JMP addr

Operation: if carry flag is set, a jump to addr is executed.

Appendix B: SX Instruction Set

JMP dest Jump to dest

Command	Words	Cycles	Affects	Coding
1) JMP addr	1	3 (8)	PC	101k kkkk kkkk JMP addr
2) JMP W	1	3 (8)	PC	0000 001f ffff JMP w
3) JMP PC+W	1	3 (8)	PC,C,DC,Z	0001 111f ffff ADD PC,W

Operation: jump to address in dest. The lower 9 bits of the literal addr is moved into the program counter in command #1. W is moved into the program counter in command #2. W+1 is added to the program counter in command #3. Bit 9 of the program counter is cleared in command #2 and #3, so the jump destination will be in the first 256 words of any 512-word page. This instruction is useful for jumping into lookup tables comprised of RETW instructions, or jumping to particular routines. The flags are set as in an ADD instruction for command #3. **If CARRYX is specified, c affects result of command #3. Insert a CLC before command #3 to avoid strange results.**

JNB src_bit, addr Jump if src_bit is not set

Command	Words	Cycles	Affects	Coding
1) JNB op.bit, addr	2	2 or 4 (jump) (8 or 12)	PC	0111 bbbf ffff SB op.bit 101k kkkk kkkk JMP addr

Operation: if src_bit is not set, a jump to addr is executed.

JNC addr Jump if carry is not set

Command	Words	Cycles	Affects	Coding
1) JNC addr	2	2 or 4 (jump) (8 or 12)	PC	0111 0000 0011 SC 101k kkkk kkkk JMP addr

Operation: if carry flag is not set, a jump to addr is executed.

JNZ addr Jump if zero is not set

Command	Words	Cycles	Affects	Coding
1) JNZ addr	2	2 or 4 (jump) (8 or 12)	PC	0111 0100 0011 SZ 101k kkkk kkkk JMP addr

Operation: if zero flag is not set, a jump to addr is executed.

--	--	--	--	--	--

Appendix B: SX Instruction Set

LJMP dest

Long jump

Command	Words	Cycles	Affects	Coding
1) LJMP addr	1 - 4	3 - 6 (8 - 20)	PC	{010x 1010 0011 CLRB/SETB 3.5} {010x 1100 0011 CLRB/SETB 3.6} {010x 1110 0011 CLRB/SETB 3.7} 101k kkkk kkkk LJMP addr

Operation: jump to a routine outside the current 512-word page. Depending on the device size, from zero to three CLRB/SETB instructions will be assembled to point the page pre-select bits to dest's page. The bit set/clear instructions are followed by a call to dest. This instruction is only useful for SXs with more than 512 words.

Note: The LJMP command cannot use the auto page-set feature, @. **This command is provided for backward compatibility only. It is suggested that the JMP @addr instruction be used instead for greater efficiency.**

LSET dest

Long set

Command	Words	Cycles	Affects	Coding
1) LSET addr	0 - 3	0 - 3 (0 - 12)	none	{010x 1010 0011 CLRB/SETB 3.5} {010x 1100 0011 CLRB/SETB 3.6} {010x 1110 0011 CLRB/SETB 3.7}

Operation: set page select bits to those in the current 512-word page referenced by dest. Depending on the device size, from zero to two CLRB/SETB instructions will be assembled. This instruction is only useful for SXs with more than 512 words.

MODE src

Set Mode to src

Command	Words	Cycles	Affects	Coding
1) MODE #literal	1	1 (4)	M	0000 0101 kkkk MOV M, #lit

Operation: src is moved into Mode register. This command is the same as MOV M, #literal. Use this command to initiate mode changes for port configuration commands.

SX Instruction Set

MOV dest, src

Move src into dest

Command	Words	Cycles	Affects	Coding
1) MOV fr, W	1	1 (4)	fr,none	0000 001f ffff MOV fr,W
2) MOV fr, #literal	2	2 (8)	fr,W	1100 kkkk kkkk MOV W,#lit 0000 001f ffff MOV fr,W
3) MOV fr1, fr2	2	2 (8)	fr1,W,Z	0010 000f ffff MOV W,fr2 0000 001f ffff MOV fr1,W
4) MOV fr, M	2	2 (8)	fr,W	0000 0100 0010 MOV W,M 0000 001f ffff MOV fr,W
5) MOV W, fr	1	1 (4)	W,Z	0010 000f ffff MOV W,fr
6) MOV W, /fr	1	1 (4)	W,Z	0010 010f ffff MOV W,/fr
7) MOV W, fr-W	1	1 (4)	W,C,DC,Z	0000 100f ffff MOV fr-W
8) MOV W, ++fr	1	1 (4)	W,Z	0010 100f ffff MOV W,++fr
9) MOV W, --fr	1	1 (4)	W,Z	0000 110f ffff MOV W,--fr
10) MOV W, <<fr	1	1 (4)	W,C	0011 010f ffff MOV W,<<fr
11) MOV W, >>fr	1	1 (4)	W,C	0011 000f ffff MOV W,>>fr
12) MOV W, <>fr	1	1 (4)	W	0011 100f ffff MOV W,<>fr
13) MOV W, #literal	1	1 (4)	W	1100 kkkk kkkk MOV W,#lit
14) MOV W, M	1	1 (4)	W	0000 0100 0010 MOV W,M
15) MOV M, fr	2	2 (8)	W,M,Z	0010 000f ffff MOV W,fr 0000 0100 0011 MOV M,W
16) MOV M, W	1	1 (4)	M	0000 0100 0011 MOV M,W
17) MOV M, #literal	1	1 (4)	M	0000 0101 kkkk MOV M,#lit
18) MOV !OPTION, fr	2	2 (8)	W,Z,OPT	0010 000f ffff MOV W,fr 0000 0000 0010 MOV !OPT,W
19) MOV !OPTION, W	1	1 (4)	OPT	0000 0000 0010 MOV !OPT,W
20) MOV !OPTION, #literal	2	2 (8)	W,OPT	1100 kkkk kkkk MOV W,#lit 0000 0000 0010 MOV !OPT,W
21) MOV !port, fr	2	2 (8)	W,Z,!port	0010 000f ffff MOV W,fr 0000 0000 0fff MOV !port,W
22) MOV !port, W	1	1 (4)	!port	0000 0000 0fff MOV !port,W
23) MOV !port, #literal	2	2 (8)	W,!port	1100 kkkk kkkk MOV W,#lit 0000 0000 0fff MOV !port,W

Operation: Src is moved into dest. Z will be set if the result is 0; otherwise, Z will be cleared. W is left holding the source value in command number 2, 3, 4, 18, 20, 21 and 23. C will be cleared if an underflow occurred; otherwise, C will be set to 1 in command number 7, 10 and 11. DC will be cleared if an underflow occurred in the lowest nibble; otherwise, DC will be set in command number 7. The value of C will be shifted into the LSB or MSB of W in command #10 and #11, respectively. C will be set to the previous MSB or LSB of fr in command #10 and #11 respectively. Command #12 moves the nibble-swapped value of fr into w. Instructions #6 through #12 are similar to NOT, SUB, INC, DEC, RL, RR and SWAP instructions, respectively, but have the additional feature of moving the result to W. **If CARRYX is specified, c affects result of command #7. Insert an STC before command #7 to avoid strange results.**

Appendix B: SX Instruction Set

MOVB dest_bit, src_bit

Move src_bit to dest_bit

Command	Words	Cycles	Affects	Coding
1) MOVB op.bit1, op.bit2	4	4 (16)	op.bit1	0111 bbbf ffff SB op.bit2 0100 bbbf ffff CLRB op.bit1 0110 bbbf ffff SNB op.bit2 0101 bbbf ffff SETB op.bit1
2) MOVB op.bit1, /op.bit2	4	4 (16)	op.bit1	0110 bbbf ffff SNB op.bit2 0100 bbbf ffff CLRB op.bit1 0111 bbbf ffff SB op.bit2 0101 bbbf ffff SETB op.bit1

Operation: src_bit is moved into dest_bit in command #1. The one's complement of src_bit is moved into dest_bit in command #2.

MOVSZ dest, src Move incremented or decremented src to dest and skip if zero

Command	Words	Cycles	Affects	Coding
1) MOVSZ W, ++fr	1	1 or 2 (skip) (4 or 8)	W	0011 110f ffff MOVSZ W, ++fr
2) MOVSZ W, --fr	1	1 or 2 (skip) (4 or 8)	W	0010 110f ffff MOVSZ W, --fr

Operation: the incremented value (command #1) or decremented value (command #2) of src is moved into dest. The next instruction word will be skipped if the result was 0.

Note: Only one word is skipped by this instruction. **To avoid strange results, make sure that any instruction following MOVSZ is a single-word instruction.**

NOP

No operation

Command	Words	Cycles	Affects	Coding
1) NOP	1	1 (4)	none	0000 0000 0000 NOP

Operation: none. This instruction is useful to adjust the timing of a routine.

SX Instruction Set

NOT dest

Not dest

Command	Words	Cycles	Affects	Coding
1) NOT fr	1	1 (4)	fr,Z	0010 011f ffff NOT fr
2) NOT W	1	1 (4)	W,Z	1111 1111 1111 NOT w

Operation: dest is converted into its one's complement value. Z will be set if the result was 0; otherwise, Z will be cleared. *The MOV W,/fr command is similar to #1, except the result is moved to W.*

OR dest, src

OR src into dest

Command	Words	Cycles	Affects	Coding
1) OR fr, W	1	1 (4)	fr,Z	0001 001f ffff OR fr,W
2) OR fr, #literal	2	2 (8)	fr,W,Z	1100 kkkk kkkk MOV W,#lit 0001 001f ffff OR fr,W
3) OR fr1, fr2	2	2 (8)	fr,W,Z	0010 000f ffff MOV W,fr2 0001 001f ffff OR fr,W
4) OR W, fr	1	1 (4)	W,Z	0001 000f ffff OR W,fr
5) OR W, #literal	1	1 (4)	W,Z	1101 kkkk kkkk OR W,#lit

Operation: src is OR'd into dest. Z will be set if the result was 0; otherwise, Z will be cleared.

PAGE addr12

Set page select bits

Command	Words	Cycles	Affects	Coding
1) PAGE addr12	1	1 (4)	none	0000 0001 0fff PAGE addr

Operation: writes address bits 11 through 9 into page select bits PA2 through PA0 in preparation for a jump or call across a page boundary.

RET

Return from subroutine

Command	Words	Cycles	Affects	Coding
1) RET	1	3 (8)	W,PC	0000 0000 1100 RET

Operation: the top stack value is moved into the program counter and execution proceeds with the instruction following the most recent call instruction.

Appendix B: SX Instruction Set

RETI

Return from interrupt routine

Command	Words	Cycles	Affects	Coding
1) RETI	1	3 (8)	C,DC,Z,PC	0000 0000 1110 RETI

Operation: W, STATUS, FSR and PC are popped off the shadow registers and execution proceeds with the instruction following the jump to interrupt.

RETIW

Return from interrupt routine, adjust RTCC

Command	Words	Cycles	Affects	Coding
1) RETIW	1	3 (8)	C,DC,Z,PC	0000 0000 1111 RETIW

Operation: W, STATUS, FSR and PC are popped off the shadow registers and execution proceeds with the instruction following the jump to interrupt. Value in W is added to RTCC. This is useful to create jitter-free, timed interrupts when using the interrupt-on-timer-rollover feature.

RETP

Return from subroutine across a page boundary

Command	Words	Cycles	Affects	Coding
1) RETP	1	3 (8)	PC	0000 0000 1101 RETP

Operation: the top stack value is moved into the program counter, return address bits 10 and 9 are written to page select bits PA1 and PA0, and execution proceeds with the instruction following the most recent call instruction. This instruction allows returning from a subroutine across page boundaries.

RETW

Assemble RETWs which load W with literal data upon return

Command	Words	Cycles	Affects	Coding
1) RETW literal {, literal...}	1 per literal	3 (8) per literal	PC	1000 kkkk kkkk RETW #lit {1000 kkkk kkkk RETW #lit...}

Operation: a list of RETWs are assembled each with one literal. This list can be accessed by JMP PC+W or JMP W instructions. This is useful for lookup tables.

SX Instruction Set

RL dest Rotate dest left

Command	Words	Cycles	Affects	Coding
1) RL fr	1	1 (4)	C	0011 011f ffff RL fr

Operation: dest is rotated left one bit. On entry, C must hold the value to be shifted into the least-significant bit of the dest value. On exit, C will hold the previous most-significant bit of the dest value. *The MOV W,<<fr command is similar to RL fr, except the result is moved to W.*

RR dest Rotate dest right

Command	Words	Cycles	Affects	Coding
1) RR fr	1	1 (4)	C	0011 001f ffff RR fr

Operation: dest is rotated right one bit. On entry, C must hold the value to be shifted into the most-significant bit of the dest value. On exit, C will hold the previous least-significant bit of the dest value. *The MOV W,>>fr command is similar to RR fr, except the result is moved to W.*

SB src_bit Skip if src_bit is set

Command	Words	Cycles	Affects	Coding
1) SB op.bit	1	1 or 2 (skip) (4 or 8)	none	0111 bbbf ffff SB op.bit

Operation: if src_bit is set, the following instruction word is skipped.

Note: Only one word is skipped by this instruction. **To avoid strange results, make sure that any instruction following SB is a single-word instruction.**

SC Skip if carry is set

Command	Words	Cycles	Affects	Coding
1) SC	1	1 or 2 (skip) (4 or 8)	none	0111 0000 0011 SC

Operation: if C is set, the following instruction word is skipped.

Note: Only one word is skipped by this instruction. **To avoid strange results, make sure that any instruction following SC is a single-word instruction.**

Appendix B: SX Instruction Set

SETB src_bit

Set src_bit

Command	Words	Cycles	Affects	Coding
1) SETB op.bit	1	1 (4)	op.bit	0101 bbbf ffff SETB op.bit

Operation: src_bit is set to 1.

SKIP

Skip the following instruction word

Command	Words	Cycles	Affects	Coding
1) SKIP	1	2 (8)	none	0110 0000 0010 SKIP

Operation: the following instruction word is skipped.

Note: Only one word is skipped by this instruction. **To avoid strange results, make sure that any instruction following SKIP is a single-word instruction.**

SLEEP

Enter sleep mode

Command	Words	Cycles	Affects	Coding
1) SLEEP	1	1 (4)	TO, PD	0000 0000 0011 SLEEP

Operation: The watchdog timer is cleared and the oscillator is stopped. TO is set and PD is cleared.

SNB src_bit

Skip if src_bit not set

Command	Words	Cycles	Affects	Coding
1) SNB op.bit	1	1 or 2 (skip) (4 or 8)	none	0110 bbbf ffff SNB op.bit

Operation: if src_bit is cleared, the following instruction word is skipped.

Note: Only one word is skipped by this instruction. **To avoid strange results, make sure that any instruction following SNB is a single-word instruction.**

SX Instruction Set

SNC

Skip if carry not set

Command	Words	Cycles	Affects	Coding
1) SNC	1	1 or 2 (skip) (4 or 8)	none	0110 0000 0011 SNC

Operation: if C is cleared, the following instruction word is skipped.

Note: Only one word is skipped by this instruction. **To avoid strange results, make sure that any instruction following SNC is a single-word instruction.**

SNZ

Skip if zero is not set

Command	Words	Cycles	Affects	Coding
1) SNZ	1	1 or 2 (skip) (4 or 8)	none	0110 0100 0011 SNZ

Operation: if Z is cleared, the following instruction word is skipped.

Note: Only one word is skipped by this instruction. **To avoid strange results, make sure that any instruction following SNZ is a single-word instruction.**

STC

Set carry flag

Command	Words	Cycles	Affects	Coding
1) STC	1	1 (4)	C	0101 0000 0011 STC

Operation: carry is set.

STZ

Set zero flag

Command	Words	Cycles	Affects	Coding
1) STZ	1	1 (4)	Z	0101 0100 0011 STZ

Operation: Z is set.

Appendix B: SX Instruction Set

SUB dest, src Subtract src from dest

Command	Words	Cycles	Affects	Coding
1) SUB fr, W	1	1 (4)	fr,C,DC,Z	0000 101f ffff SUB fr,W
2) SUB fr1, #literal	2	2 (8)	fr,W,C,DC,Z	1100 kkkk kkkk MOV W,#lit 0000 101f ffff SUB fr,W
3) SUB fr1, fr2	2	2 (8)	fr,W,C,DC,Z	0010 000f ffff MOV W,fr 0000 101f ffff SUB fr,W

Operation: src is subtracted from dest. C will be cleared to 0 if an underflow occurred; otherwise, C will be set to 1. DC will be cleared to 0 if an underflow occurred in the least-significant nibble. Z will be set to 1 if the result was 0; otherwise, Z will be cleared to 0. *The MOV W,fr-W command is similar to #1, except the result is moved to W. If CARRYX is specified, c is added to result. Insert an STC before the first Sub on a register to avoid strange results.*

SUBB dest, src_bit Subtract src_bit from dest

Command	Words	Cycles	Affects	Coding
1) SUBB fr, op.bit	2	2 (8)	Z	0110 bbbf ffff SNB op.bit 0000 111f ffff DEC fr
1) SUBB fr, /op.bit	2	2 (8)	Z	0111 bbbf ffff SB op.bit 0000 111f ffff DEC fr

Operation: subtract src_bit from dest. If dest was decremented, Z will be set if the result was zero; else, Z will be cleared. This instruction is useful for subtracting the carry from the upper byte of a double-byte value after the lower byte has been subtracted.

SWAP dest Swap nibbles in dest

Command	Words	Cycles	Affects	Coding
1) SWAP fr	1	1 (4)	none	0011 101f ffff SWAP fr

Operation: The high- and low-order nibbles of dest are swapped. *The MOV W,<>fr command is similar to SWAP fr, except the result is moved to W.*

SX Instruction Set

SZ

Skip if zero flag set

Command	Words	Cycles	Affects	Coding
1) SZ	1	1 or 2 (skip) (4 or 8)	none	0111 0100 0011 SZ

Operation: if Z is set, the following instruction word is skipped.

Note: Only one word is skipped by this instruction. **To avoid strange results, make sure that any instruction following SZ is a single-word instruction.**

TEST src

Test src for zero

Command	Words	Cycles	Affects	Coding
1) TEST fr	1	1 (4)	Z	0010 001f ffff TEST fr
2) TEST w	1	1 (4)	Z	1101 0000 0000 TEST w

Operation: Z will be set if src is 0; otherwise, Z will be cleared.

XOR dest, src

XOR src into dest

Command	Words	Cycles	Affects	Coding
1) XOR fr, W	1	1 (4)	fr,Z	0001 101f ffff XOR fr,w
2) XOR fr, #literal	2	2 (8)	fr,W,Z	1100 kkkk kkkk MOV w,#lit 0001 101f ffff XOR fr,w
3) XOR fr1, fr2	2	2 (8)	fr,W,Z	0010 000f ffff MOV w,fr2 0001 101f ffff XOR fr1,w
4) XOR W, fr	1	1 (4)	W,Z	0001 100f ffff XOR w,fr
5) XOR W, #literal	1	1 (4)	W,Z	1111 kkkk kkkk XOR w,#lit

Operation: src is XOR'd into dest. Z will be set if the result was zero; otherwise, Z will be cleared.

Appendix C: SX-Key Reserved Words

ADD	FSR	OSC1MHZ	RETP
ADDB	FSTR	OSC125KHZ	RETW
AND	HS_OSC	OSC2MHZ	RF
BANK	ID	OSC250KHZ	RG
BANKS1	IF	OSC31KHZ	RL
BANKS2	IFDEF	OSC4MHZ	RR
BANKS4	IFNDEF	OSC500KHZ	RTCC
BANKS8	IJNZ	OSC62KHZ	SB
BROWNOUT	INC	OSCHS	SBIN
BREAK	INCSZ	OSCXT	SDEC
C	INDF	OSCLP	SC
CALL	INDIRECT	OSCRC	SETB
CARRYX	IND	PA0	SHEX
CASE	IRCDIV1	PA1	SKIP
CJA	IRCDIV2	PA2	SLEEP
CJAE	IRCDIV4	PAGE	SNB
CJB	IRCDIV8	PAGES1	SNC
CJBE	IRCDIV16	PAGES2	SNZ
CJE	IRCDIV32	PAGES4	STACKX
CJNE	IRCDIV64	PC	STATUS
CLC	IRCDIV128	PCL	STC
CLR	IREAD	PD	STZ
CLRB	JB	PIC16C54	SUB
CLZ	JC	PIC16C54A	SUBB
CSA	JMP	PIC16C55	SWAP
CSAE	JNB	PIC16C56	SYNC
CSB	JNC	PIC16C57	SZ
CSBE	JNZ	PIC16C58	TEST
CSE	JZ	PIC16C58A	TMR0
CSNE	LCALL	PINS18	TO
DC	LJMP	PINS28	TURBO
DEC	LP_OSC	PORT_B	UBIN
DECSZ	LSET	PORT_C	UDEC
DEVICE	M	PROTECT	UHEX
DJNZ	MACRO	PROTECT_OFF	W
DS	MODE	PROTECT_ON	WATCH
DW	MOV	RA	WATCHDOG
ELSE	MOVB	RB	WDT
END	MOVSZ	RC	WDT_OFF
ENDIF	NOCASE	RC_OSC	WDT_ON
ENDM	NOEXPAND	RD	WREG
ENDR	NOP	RE	XOR
EQU	NOT	REPT	XT_OSC
ERROR	OPTION	RESET	Z
EXITM	OPTIONX	RET	ZSTR
EXPAND	OR	RETI	
FREQ	ORG	RETIW	

SX-Key Reserved Words

Appendix D: SX-Key Error Messages

SX-Key Error Messages

This appendix lists the SX-Key Assembler error messages. Most error messages below are followed by a short explanation and troubleshooting tips.

- **“=” must be preceded by a variable**
 - No valid symbol exists to the left of the “=”. Check for mistyped symbol. Watch out for different case when the CASE directive is used.
 - Make sure symbol does not appear in Reserved Words list in Appendix C.
- **“\” only allowed in MACRO definition**
 - The macro argument symbol, “\”, is a meaningless character outside of macros.
- **CALL must be to first half of page**
 - The destination address of the CALL instruction points to the second half of a page. See Calling Across Pages in Chapter 7 for more information.
- **Constant exceeds 32 bits**
 - The SX-Key assembler can not handle constants whose value is larger than 32 bits or 64 digits.
- **Constant exceeds 64 digits**
 - The SX-Key assembler can not handle constants whose value is larger than 32 bits or 64 digits.
- **Clock frequency must be from 400_000 to 110_000_000**
 - Designated frequency used in the FREQ directive is outside the range. The SX-Key can only clock the SX chip between 400 KHz and 110 KHz.
- **ELSE/ENDIF must be preceded by IF**
 - Check for missing or commented-out IF directive above the ELSE/ENDIF.

SX-Key Error Messages

- **Empty string**
 - Look for undefined string within apostrophes.
- **ENDIF required to end IF block**
 - Check for missing or commented-out ENDF directive below the IF.
- **ENDM required to end MACRO definition**
 - Check for missing or commented-out ENDM at the end of a MACRO definition.
- **ENDR must be preceded by REPT**
 - Check for missing or commented-out REPT directive above the ENDR.
- **ENDR required to end REPT block**
 - Check for missing or commented-out ENDR directive below the REPT.
- **EQU must be preceded by a label**
 - A valid symbol must precede the EQU directive. Check for mistyped symbol. Watch out for different case when the CASE directive is used.
 - Make sure symbol does not appear in Reserved Words list in Appendix C.
- **Error message contains control characters**
 - Error message must contain printable characters only.
- **Error message exceeds 64 characters**
 - Error message must be 64 characters or less in length.
- **EXITM/ENDM must be preceded by MACRO**
 - Check for missing or commented-out MACRO directive above the EXITM/ENDM.
- **Expected “++” or “--”**
 - The source operand in a MOVSZ must be preceded by ++ or --.

Appendix D: SX-Key Error Messages

- **Expected “,”**
 - Check for missing arguments on a multi-argument mnemonic.
- **Expected “,” or end-of-line**
 - Check for invalid character(s) at the end of the line.
- **Expected a binary operator or “)”**
- **Expected a constant**
- **Expected a constant, variable, unary operator, or “(“**
 - Look for mnemonic with bad or missing arguments. Look for incomplete expressions.
 - Make sure symbol in use does not appear in Reserved Words list in Appendix C.
- **Expected a DEVICE parameter**
 - DEVICE directive contains a missing or invalid parameter. Look for misspellings, commas without trailing parameters, lower case when using CASE directive, etc.
- **Expected a label**
- **Expected a label, directive, or instruction**
 - Check for invalid arguments.
 - Check for mistyped mnemonic.
- **Expected a value from 0 to 64 or end-of-line**
 - Argument count on macros must be 0 to 64 or not specified.
- **Expected a value from 1 to 32**
 - The count parameter in the WATCH directive must be in the range of 1 to 32.
- **Expected a terminating quote**
 - Look for a string without a terminating quote, or apostrophe.
- **Expected an expression**
 - Look for a mnemonic with missing arguments.

SX-Key Error Messages

- **Expected end-of-line**
 - Look for invalid character or argument at the end-of-line.
 - Look for incomplete expression.
- **Expected UDEC, SDEC, UHEX, SHEX, UBIN, SBIN, FSTR or ZSTR**
 - WATCH directive is missing formatter argument.
- **Expected W**
 - The W argument is missing in a mnemonic that requires it.
- **Expected WDT**
 - The clear-watchdog mnemonic must specify !WDT.
- **Expression is too complex**
 - SX-Key assembler can not handle the designated expression. Try simplifying the expression if possible.
- **ID cannot exceed 8 characters**
 - A maximum of 8 characters are allowed in the ID directive.
- **ID must be a string of up to 8 characters**
 - Make sure to use single quotes, or apostrophes, ('), before and after the string. Make sure not to input control characters.
- **Location already contains data**
 - Assembled instruction overlaps used memory or crossed over last defined page barrier. Can also occur when the RESET directive is specified twice.
- **Label is already defined**
 - A symbol, or label, is already defined or is a reserved word. See the Reserved Words list in Appendix C. Make sure label's position is not invalid, such as a label in a REPT block (this would make duplicate labels during the expansion). Make sure label starts with a letter or an underscore (_).

Appendix D: SX-Key Error Messages

- **Limit of 32 nested REPTs exceeded**
 - The SX-Key assembler cannot process more than 32 nested REPT blocks.
- **Limit of 100,000 total REPT loops exceeded**
 - REPT count argument must be in the range 1..100,000.
- **List is too large**
 - List file generation can not complete because it is too large. Look for REPT blocks whose count is high, or whose final size, during assembly, is large.
- **Macro argument index is out of range**
 - The designated argument index is outside the specified range as set by the macro's definition.
- **Macro argument is not resolvable**
- **MACRO must be preceded by a label**
 - A valid symbol must precede the MACRO directive. Check for mistyped symbol.
 - Make sure symbol does not appear in Reserved Words list in Appendix C.
- **Macro stack overflow**
 - Macro is too complex; try simplifying it.
- **Nothing to assemble**
 - Must have source code entered or loaded up into the editor before assembling, programming, running or debugging.
- **Only one BREAK is allowed**
 - The SX chip does not support more than one breakpoint at a time.
- **Port is out of range**
 - Verify the port symbol or address in the mnemonic. See Appendix F for available ports.

SX-Key Error Messages

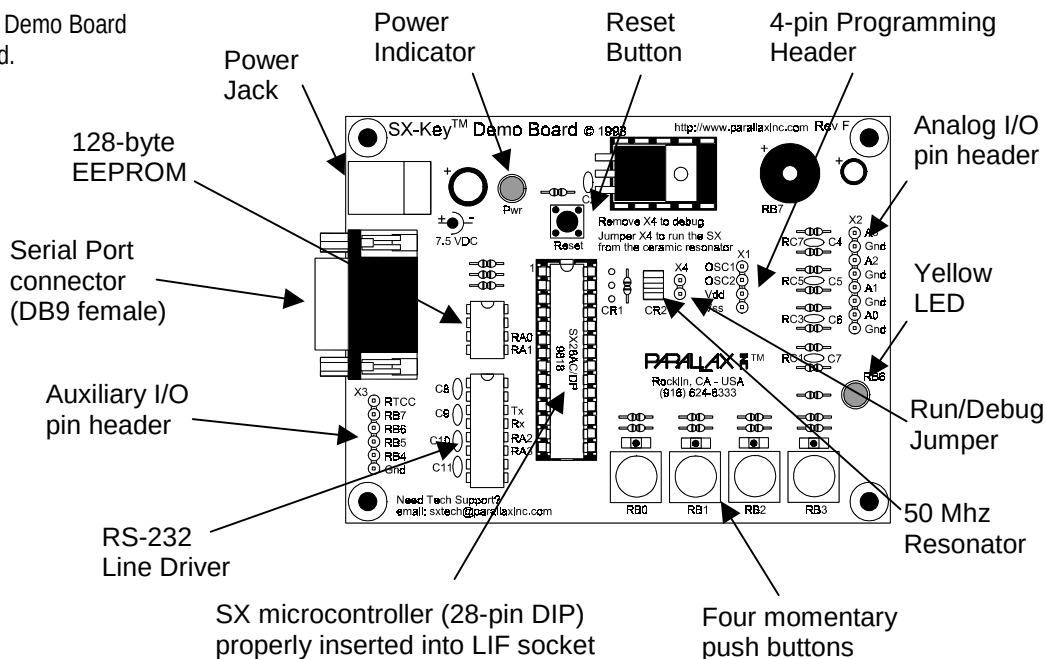
- **Redundant DEVICE parameter**
 - The parameter specified conflicts with a previously specified device parameter.
- **REPT count must be greater than 0**
- **RESET address must be on first page**
 - The SX chip does not support a reset address outside of page 0.
- **Symbol exceeds 32 characters**
 - All symbols must be 32 characters in length or less and must start with a letter or underscore (_).
- **Symbol table full**
 - Too many symbols are defined. Must limit or combine any applicable symbols to assemble properly.
- **This directive cannot be preceded by a symbol**
 - Only the ORG, RESET, EQU, =, DS, DW, BREAK, MACRO and END directives can be preceded by a symbol.
- **Undefined Symbol**
 - Symbol is not defined above highlighted line. Check for mistyped symbol. Watch out for different case when the CASE directive is used.
 - Make sure symbol does not appear in Reserved Words list in Appendix C.
- **Unrecognized character**
 - Use single quotes (') instead of double quotes (").
- **Variable must be followed by “=”**
 - Look for mistyped label.

Appendix E: SX-Key Demo/QuickProto Boards

Introduction

The Parallax SX-Key Demonstration Boards and the SX-Key QuickProto Boards are learning tools for 28-pin and 18-pin DIP SX microcontrollers, respectively. The `sxdemo.src` source code, included on the Parallax SX-Key Development System diskette, demonstrates Virtual Peripherals on the SX-Key Demo Board.

Figure E.1: SX-Key Demo Board with SX chip inserted.



SX-Key Demo Board Features

The SX-Key Demo Board contains many pre-wired components to make development with the 28-pin SX DIP microcontroller easier. **The SX-Key Demo Board does not support the 18-pin SX chip; please see the SX-Key QuickProto Board section, below, for more information.**

SX-Key Demo/QuickProto Boards

Features

The SX-Key Demo Board includes the following items:

- 7.5 VDC 1 A, center positive, power supply input;
- Power indicator LED;
- 28-pin LIF socket;
- 50 MHz ceramic resonator;
- Jumper to enable / disable on-board resonator;
- (4) momentary pushbuttons;
- (1) yellow and (4) red LEDs to indicate status of buttons or I/O pins;
- 32 Ohm, 500 Hz – 8 KHz speaker;
- (2) digital to analog output pins;
- (2) analog to digital input pins;
- Auxiliary I/O pin header;
- RS-232 driver and DB9 (female) connector;
- 128 byte EEPROM;
- Reset button.

Virtual Peripheral Source Code Example

The `sxdemo.src` file on the SX-Key Development System diskette works with the SX-Key Demo Board to demonstrate Virtual Peripherals. The Virtual Peripherals demonstrated by `sxdemo.src` are:

- One 19.2k baud UART for full-duplex communication between PC and the SX demo board over a standard RS-232 serial cable.
- Two 16-bit timers for controlling LED blink rate and speaker frequency output.
- Two 8-bit PWM/DAC outputs.
- Two ADC inputs.

Connecting and Downloading

See Chapter 2 for steps to connect and use the demo board with the `sxdemo.src` program.

Appendix E: SX-Key Demo/QuickProto Boards

Communication

The UART virtual peripheral allows communication between your PC and the SX chip on the demonstration board at 19.2k baud. See Chapter 2 for setup information.

Once the HyperTerminal connection is running, and the sxdemo.src program has already been downloaded to the SX in the demo board, press and release the Reset button on the demo board. You should hear a clicking noise, the yellow LED should flash and “SX Virtual Peripheral Demo” should appear on your screen.

Your cursor should appear next to a “>” prompt. Now you can type in simple commands to change the behavior of the virtual peripherals.

Configuring the Virtual Peripherals

Commands entered within HyperTerminal (or other properly configured terminal program) will modify the parameters used in each virtual peripheral. The SX is programmed to execute the virtual peripherals simultaneously. Table E.1 indicates the available commands and their function.

Table E.1: Virtual Peripheral Demo Commands.

Command	Description
Fxxx	Uses an independent 16-bit timer to control the output frequency / sound of the speaker. Parameter xxx is a number between 0 and FFF hex. Only low numbers are audible tones.
Txxx	Controls LED blinking rate. Parameter xxx is a number between 0 and FFF hex. Values between 0 and 6 display a noticeable LED flash and values up to FFF hex appear to make the LED continually stay “on” (though it is actually blinking).
Axx	Controls channel 1 PWM output on pin A0. Parameter xx is a number between 00 and FF hex, which will generate an analog voltage between 0 and 5 V. A simple RC circuit is attached to the I/O pin to filter the PWM.
Bxx	Controls channel 2 PWM output on pin A1. Parameter xx is a number between 00 and FF hex, which will generate an analog voltage between 0 and 5 V. A simple RC circuit is attached to the I/O pin to filter the PWM.
C	Channel 1 ADC input on pin A2. Using the capacitor on the A2 analog input pin, the SX reads voltage values on this pin and represents 0V to 5V as 00 to FF hex, respectively.
D	Channel 2 ADC input on pin A3. Using the capacitor on the A3 analog input pin, the SX reads voltage values on this pin and represents 0V to 5V as 00 to FF hex, respectively.

SX-Key Demo Board Schematic

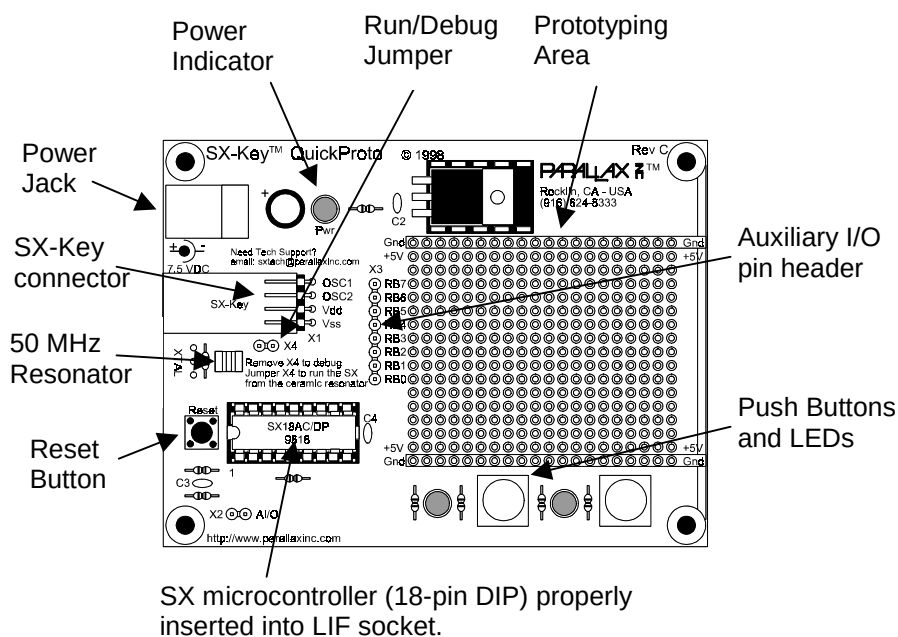


Appendix E: SX-Key Demo/QuickProto Boards

SX-Key QuickProto Board Features

The SX-Key QuickProto Board contains pre-wired components and through-hole prototyping space to make development with the 18-pin SX DIP microcontroller easier. **The SX-Key QuickProto Board does not support the 28-pin SX chip; please see the SX-Key Demo Board section, above, for more information.**

Figure E.2: SX-Key QuickProto Board.



SX-Key Demo/QuickProto Boards

Features

The SX-Key QuickProto Board includes the following items:

- 7.5 VDC 1 A, center positive, power supply input;
- Power indicator LED;
- 18-pin LIF socket;
- 50 MHz ceramic resonator;
- Jumper to enable / disable on-board resonator;
- (2) momentary pushbuttons;
- (2) LEDs to indicate status of buttons or I/O pins;
- (1) analog I/O pin;
- Auxiliary I/O pin header;
- 3.4 square inches of through-hole prototyping area;
- Reset button.

Connecting and Downloading

See Chapter 2 for steps to connect and use the QuickProto board with the led18.src program.

SX-Key QuickProto Board Schematic



SX-Key Demo/QuickProto Boards

Appendix F: SX Data Sheet

Pinout Information and Descriptions

Figure F.1: SX Pinouts.

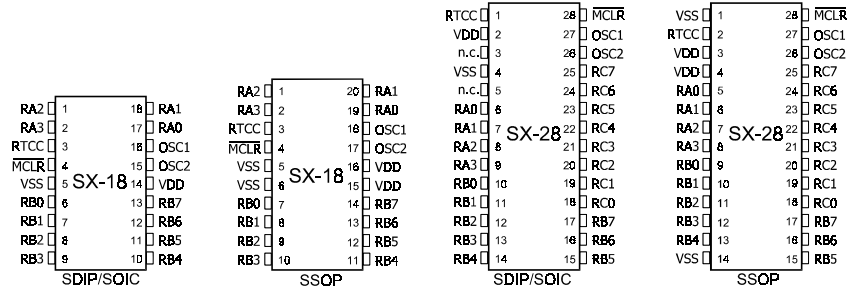


Table F.1: SX Pinout Descriptions.

Name	Pin Type	Input Levels	Description
RA0 - RA3	I/O	TTL/CMOS	Bi-directional I/O Pin, Complimentary Drive
RB0 - RB2	I/O	TTL/CMOS/ST	Bi-directional I/O Pin; MIWU mode; Comparator output, - input, + input
RB3 - RB7	I/O	TTL/CMOS/ST	Bi-directional I/O Pin; MIWU mode
RC0 - RC7	I/O	TTL/CMOS/ST	Bi-directional I/O Pin
RTCC	I	ST	Input to Real Time Clock/Counter
MCLR	I	ST	Master Clear (reset) input. This pin is an active low reset to the device.
OSC1	I	ST	Oscillator crystal input - external clock source input.
OSC2	I/O	CMOS	Weakly pulled to Vdd internally on RC mode.
Vdd	P	-	Positive supply for logic and I/O pins.
Vss	P	-	Ground Reference for logic and I/O pins.

Note: I = input, O = output, I/O = Input/Output, P = Power, - = Not Applicable, TTL = TTL input, CMOS = CMOS input, ST = Schmitt Trigger input, MIWU = Multi-Input Wake Up

SX Data Sheet

	Bank 0	Bank 1	Bank 2	Bank 3	Bank 4	Bank 5	Bank 6	Bank 7
\$00	IND	IND	IND	IND	IND	IND	IND	IND
\$01	RTCC	RTCC	RTCC	RTCC	RTCC	RTCC	RTCC	RTCC
\$02	PC	PC	PC	PC	PC	PC	PC	PC
\$03	Status	Status	Status	Status	Status	Status	Status	Status
\$04	FSR	FSR	FSR	FSR	FSR	FSR	FSR	FSR
\$05	Port A	Port A	Port A	Port A	Port A	Port A	Port A	Port A
\$06	Port B	Port B	Port B	Port B	Port B	Port B	Port B	Port B
\$07	Port C	Port C	Port C	Port C	Port C	Port C	Port C	Port C
\$08	8 General Purpose RAM registers (\$08 - \$0F)							
\$09								
\$0A								
\$0B								
\$0C								
\$0D								
\$0E								
\$0F								
\$10	16 General Purpose RAM Registers (\$10 - \$1F)	16 General Purpose RAM Registers (\$30 - \$3F)	16 General Purpose RAM Registers (\$50 - \$5F)	16 General Purpose RAM Registers (\$70 - \$7F)	16 General Purpose RAM Registers (\$90 - \$9F)	16 General Purpose RAM Registers (\$B0 - \$BF)	16 General Purpose RAM Registers (\$D0 - \$DF)	16 General Purpose RAM Registers (\$F0 - \$FF)
\$11								
\$12								
\$13								
\$14								
\$15								
\$16								
\$17								
\$18								
\$19								
\$1A								
\$1B								
\$1C								
\$1D								
\$1E								
\$1F								

Figure F.2: RAM Register Map.

Register Map Structure

The SX RAM memory consists of special function registers and general purpose registers organized into as many as eight banks of 32 locations. Figure F.2 demonstrates the structure of the registers. The upper 16 registers (\$00 - \$0F) are not banked while the lower 16 (\$10 - \$1F) are banked. For example, reading location \$03 of Bank 5 is the same as reading location \$03 of Bank 0. However, location \$1A of Bank 5 is a unique, general purpose RAM register which is not the same as location \$1A in Bank 0.

Architecture

The Scenix SX chip offers 2K x 12 internal EE/Flash program memory and up to 137 bytes of RAM memory. The EE/Flash memory is organized in 512-word pages. The RAM memory is addressable directly or indirectly. All special function registers are mapped into the

Appendix F: SX Data Sheet

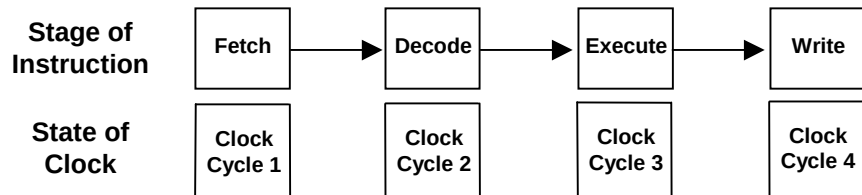
data memory. Configuration registers do not appear in data memory and are only accessible through the use of the MODE register and the port configuration commands.

The ALU is 8-bits wide and is capable of arithmetic and Boolean operations. The 'W' register is the working register for the ALU. Typically, it holds one operand in a two-operand instruction. Depending on the instruction executed, the ALU may affect the values of the Carry (C), Zero (Z), and Digit Carry (DC) flags of the STATUS register.

The SX chip comes equipped with special features that reduce system cost and power requirements. The Power-On Reset (POR) and Device Reset Timer eliminate the need for external reset circuitry. The power saving SLEEP mode, watchdog timer, and code protect features reduce system cost and improve system integrity.

Instruction Pipeline

Figure F.3: Instruction Pipeline.



There are several stages an instruction must go through to actually execute within the SX chip. Specifically, there are four stages that are collectively referred to as the pipeline, and are shown in Figure F.3. The first instruction is fetched from memory on the first clock cycle. On the second clock cycle the first instruction is decoded and the second instruction is fetched. On the third clock cycle the first instruction is executed, the second instruction is decoded, and the third instruction is fetched. On the fourth clock cycle the first instruction's results are written to its destination, the second instruction is executed, the third instruction is decoded and the fourth instruction is fetched. Once the pipeline is full, instructions are executed at the rate of one per clock cycle (in Turbo mode). Instructions that directly alter the value in the program counter, i.e. jumps, calls, etc. require that the pipeline be

SX Data Sheet

cleared and subsequently refilled. When the pipeline is cleared, the fetch and decode stages are replaced with ‘nop’ instructions. This effectively nullifies the invalid instructions.

Read-Modify-Write Considerations

Use caution when performing successive SETB or CLRB operations on an I/O port pin. Since input data used for an instruction must be valid *during* the time the instruction is executed, and the result output from an instruction is valid *after* that instruction completes its operation, unexpected results from successive read-modify-write operations on I/O pins can occur when the SX is running at extremely high speeds. The SX has an internal write-back section to prevent such data errors from occurring but it is recommended that you buffer successive read-modify-write instructions performed on I/O pins of the same port at extremely high clock rates with a ‘nop’ instruction.

Also note, a read of an I/O pin actually reads the pin, not the output data latch. That is, if an output driver on a pin is enabled and driven high, but the external circuit is holding it low, a read of the port pin will indicate that the pin is low. Of course, externally driving an I/O pin while the output latch is driving it will result in damage to the SX chip. Care should be taken to not do this.

Special Function Registers

Special function registers are registers used by the CPU to control the operation of the device.

Addr	Name	Function
\$00	IND	Used for indirect addressing
\$01	RTCC / WREG	Real Time Clock Counter / WREG
\$02	PC	Program Counter (low byte)
\$03	STATUS	Holds Status bits of ALU
\$04	FSR	File Select Register
\$05	RA	Port A Control register
\$06	RB	Port B Control register
\$07	RC*	Port C Control register

*Available as general RAM in the SX18

Table F.2: Special Function Registers.

Appendix F: SX Data Sheet

IND - The Indirect Register (\$00)

This register, though not physically implemented, is used for indirect addressing. An instruction using IND as its operand actually performs the operation on the register pointed to by the contents of FSR.

Direct and Indirect Addressing Examples

Direct addressing:

```
mov    RA,#01        ;move '1' to RA
```

Indirect addressing:

```
mov    FSR,#RA        ;FSR = address of RA
mov    IND,#01         ;move '1' to RA
```

Note that with indirect addressing, any operation on IND results in the operation being performed on the register *pointed to* by the FSR. Refer to FSR - File Select Register for more information.

RTCC Real Time Clock/Counter and WREG (\$01)

RTCC is an 8-bit real-time timer/counter. In timer mode, the RTCC register will increment with every instruction cycle (without prescaler). In counter mode, the RTCC will increment with every cycle on the RTCC pin (with prescaler). The prescaler is used to lengthen the RTCC or watchdog timer effectively up to 16-bits. Depending on the RTW bit (OPTION.7), register \$01 contains either the RTCC, (RTW is set) or the WREG, (RTW cleared). When WREG exists at \$01, file register instructions (INC, DECSZ, etc) can be used directly on WREG. When doing this, use the register address \$01, or the WREG symbol, rather than W. Using the W symbol instead of WREG to operate directly on the working register will result in errors or incorrectly assembled code.

PC - Program Counter (\$02)

PC is a register that holds the lower 8-bits of the program counter. It is accessible at runtime to perform computed jumps and determine return addresses. Whenever an instruction is executed, and PC is the destination, the upper bits of the STATUS register are loaded into the high byte of the program counter. This is necessary to achieve computed jumps and subroutine calls *across* page boundaries in code memory.

SX Data Sheet

The Program Counter

The program counter contains the 12-bit address (code memory) of the instruction to be executed. Only 11 bits are used in current SX parts. Below are examples detailing how the program counter and the STATUS register are used in typical situations.

Jump Instruction

When a JMP instruction is executed, the lower nine bits of the program counter are loaded with the address of the label specified. The upper two bits of the program counter are loaded with the page select bits, PA1:PA0, from the STATUS register. Therefore, care must be used to ensure the page select bits are pointing to the correct page *before* the jump occurs.

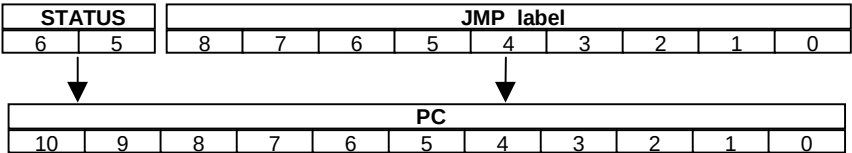


Figure F.4: Jump Instruction.

Jumping Across Pages

When a JMP instruction is executed and the intended destination is on a different page, you must set the page select bits to point to the desired page before the jump occurs. This can be done discretely with SETB and CLRB instructions or by writing a value to the STATUS register. The SX offers a new single-cycle instruction called PAGE that sets the page select bits for you. The example below shows how to use PAGE with the JMP instruction. *NOTE: Using the @ symbol in the JMP instruction (JMP @label) will cause the SX-Key assembler to insert the PAGE instruction before your JMP, during assembly.*

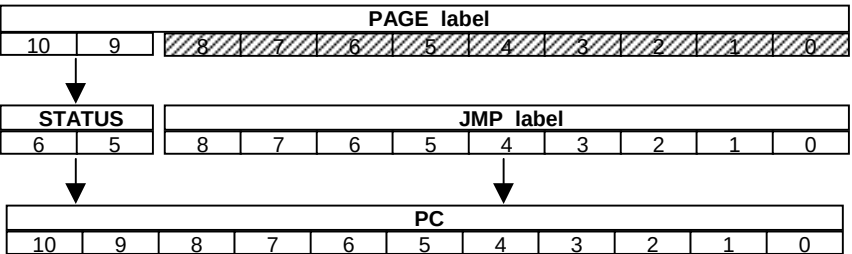


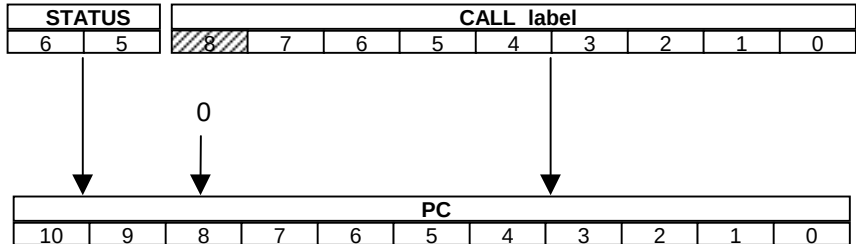
Figure F.5: Page-Jump Instructions.

Appendix F: SX Data Sheet

Call Instruction

When a CALL instruction is executed, four things occur: 1) the current value of the program counter is incremented and pushed onto the top of the stack; 2) the lower eight bits of the address of the label are copied into the lower eight bits of the program counter; 3) the ninth bit of the PC is cleared to zero; and 4) the page select bits of the STATUS register are copied into the upper bits of the PC. Since bit 8 is cleared, the call destination must *start* in the lower half of any page of code space. i.e. \$00-\$FF, \$200-\$2FF, \$400-\$4FF, etc.

Figure F.6: Call Instruction.



Calling Across Pages

When it is necessary to call a subroutine that exists on a different page, you must set the page select bits to point to the desired page before the call is executed. This can be done discretely using SETB and CLRB instructions or by writing a value to the STATUS register. The SX offers a new single-cycle instruction called PAGE that sets the page select bits for you. The example below shows how to use PAGE with the CALL instruction. *NOTE: Using the @ symbol in the CALL instruction (CALL @label) will cause the SX-Key assembler to insert the PAGE instruction before your CALL, during assembly.*

SX Data Sheet

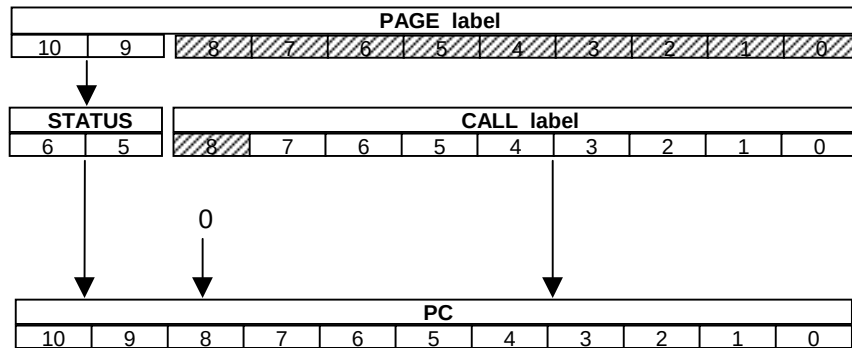


Figure F.7: Page-Call Instructions.

Returning from a subroutine

Subroutines are usually terminated with a return type instruction. Before we discuss the different return instructions, we should describe the operation of the stack.

The Stack

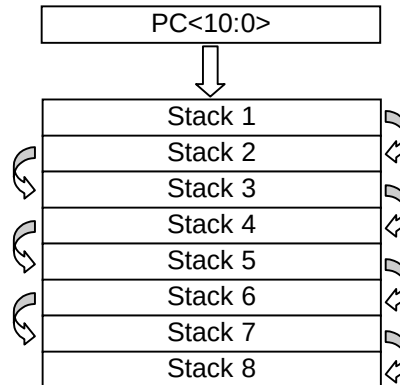
The stack is an area of memory used to remember where to return to once a subroutine is complete. The stack is eight levels deep with the Stack Extend option set and two levels deep by default. That means it can remember the return addresses for subroutines nested up to eight levels. The following explanation assumes that the SX has the Stack Extend option selected. The stack is capable of two operations; push and pop. The stack behaves like a plate holder in a salad buffet. A push is similar to placing a plate on the top of the stack and a pop is similar to removing a plate from the top of the stack.

The Push

When a subroutine is called, the return address is pushed onto the stack. Specifically, each address in the stack is moved to the next lower level in order to make room for the new address to be stored. Stack 1 gets the value that was in the program counter. Stack 8 is overwritten with what was in Stack 7. Consequently, the previous contents of Stack 8 are lost forever.

Appendix F: SX Data Sheet

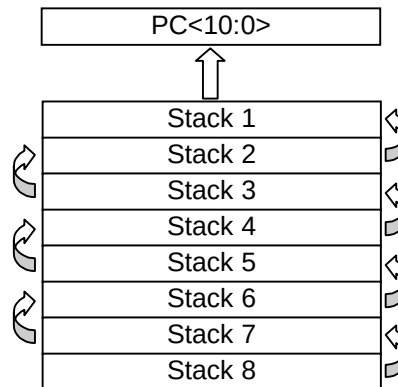
Figure F.8: Stack Push Operation.



The Pop

When a return instruction is executed, the stack is popped. Specifically, the content of Stack 1 is copied to the program counter and the content of each address in the stack is copied to the next higher level. Stack 1 gets the value that was in Stack 2, etc. until Stack 7 is overwritten with the contents of Stack 8. Consequently, the contents that were in Stack 8 are now duplicated in Stack 7.

Figure F.9: Stack Pop Operation.



Stack Overflow

As mentioned before, the stack can store up to eight return addresses (with Stack Extend on), or up to two return addresses by default. With each push, the stack stores another address. When the stack is full, the next push results in an overflow. The first time the stack is pushed into an overflow condition, the first address pushed is lost forever. If the

SX Data Sheet

stack were to be pushed again, the second address pushed would be lost also. A stack overflow condition inevitably leads to unintentional infinite loops or bizarre looping actions in your program. Care should be taken to ensure a stack overflow does not ever occur.

Stack Underflow

When the stack is popped more times than it has been pushed, a stack underflow occurs. Since a stack underflow causes unknown addresses to be stored in PC, a program may perform bizarre looping actions; such as a jump to unused program memory.

Returns

There are five different return instructions available on the SX. The RET (return) instruction simply pops the stack thereby setting the program counter to the instruction that followed the call. The RETW (return with literal in w) instruction behaves the same way but loads W with the literal value specified. RETP pops the stack and updates the page select bits to point to the page returned to. RETI pops the stack and the special shadow registers for W, STATUS, and the FSR, which were preserved during interrupt handling. RETIW behaves the same as RETI but also compensates the RTCC by adding the value in W to the RTCC.

STATUS Register (\$03)

This register holds the arithmetic status of the ALU, the page select bits, and the reset status. The status register is accessible during run-time but the resets bits, PD and TO, are read only. Care should be used when writing to the STATUS register as the ALU status bits will be updated upon completion of the write operation thereby leaving the STATUS register with a result that is different than intended. Therefore, it is recommended that only SETB, CLRB and PAGE instructions be used on this register.

Appendix F: SX Data Sheet

Figure F.10: Status Register.

STATUS							
7	6	5	4	3	2	1	0
PA2	PA1	PA0	TO	PD	Z	DC	C

Bits 7-5: *Page select bits (PA2:PA0)

000 = Page 0 (\$000 – \$1FF)

001 = Page 1 (\$200 – \$3FF)

010 = Page 2 (\$400 – \$5FF)

011 = Page 3 (\$600 – \$7FF)

100 = Page 4 (\$800 – \$9FF)

101 = Page 5 (\$A00 – \$BFF)

110 = Page 6 (\$C00 – \$DFF)

111 = Page 7 (\$E00 – \$FFF)

* For devices of less than 4K of code space, unused bits of PA2:PA0 may be used as general purpose read / write bits.

Bit 4: Time Out bit (TO).

1 = Set to '1' after power up, or executing a CLR !WDT or SLEEP instruction.

0 = A watchdog time-out occurred.

Bit 3: Power Down bit (PD).

1= Set to a '1' after power up or executing a CLR !WDT instruction.

0 = Set to a '0' by a SLEEP instruction

Bit 2: Zero Bit (Z).

1 = Result of math operation is zero

0 = Result of math operation is non-zero

Bit 1: Digit Carry (DC).

After Addition:

1 = A carry from bit-4 occurred

0 = No carry from bit-4 occurred

After Subtraction:

1 = No borrow from bit-4 occurred

0 = A borrow from bit-4 occurred

Bit 0: Carry (C).

After Addition:

1 = A carry occurred

0 = No carry occurred

After Subtraction:

1 = No borrow occurred

0 = A borrow occurred

The carry flag also serves as the ninth bit in RL and RR instructions. We can examine the operation of each of these instructions to further clarify the behavior of the carry flag. Consider the RR instruction first. When

SX Data Sheet

an RR instruction is performed on a RAM byte, the data in the RAM byte is rotated *through* the carry flag.

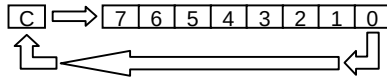


Figure F.11: Rotate-Right Operation.

Similarly, when an RL instruction is performed on a RAM byte, the data in the RAM byte is rotated through the carry flag.

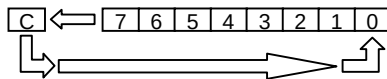


Figure F.12: Rotate-Left Operation.

FSR - File Select Register (\$04)

The Scenix SX chip utilizes 12-bit op-codes. Instructions that specify a register as an operand can only express 5-bits of register address. This means that only registers from \$00 up to \$1F can be accessed. The File Select Register (FSR) is used to provide the ability to access registers beyond \$1F. Figure F.13 shows how the FSR's upper three bits select one of eight RAM banks.

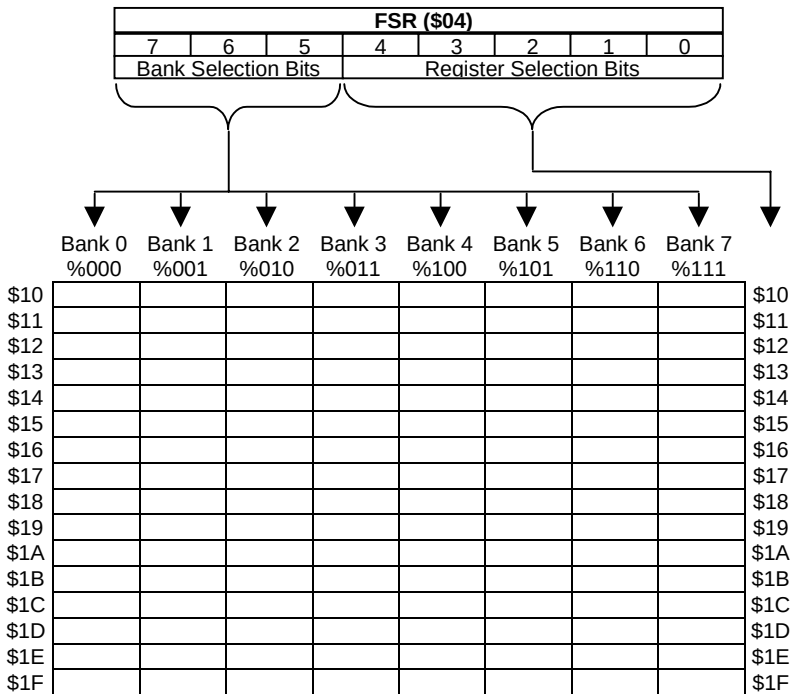
Special function and general purpose register addresses \$00 - \$0F are 'global' in that they can always be accessed regardless of the contents of the FSR. Special function register \$07 (RC) is available as general purpose RAM in 18-pin SX packages.

Here's an example of how to write to register \$10 on bank 4:

```
mov FSR,$90      ;Select Bank 4 by setting FSR<7:5>
mov $10,$64      ;load register $10 with the literal $64
```

Appendix F: SX Data Sheet

Figure F.13: File Select Register and RAM relationship.



Direct Addressing

To ensure you are writing to the desired register you must first write the correct value to the FSR. The following table and code fragment will show you how to directly access the banked registers.

Table F.3: Banks and FSR values.

If you want bank...	Value to move to FSR is...
0	\$00
1	\$30
2	\$50
3	\$70
4	\$90
5	\$B0
6	\$D0
7	\$F0

SX Data Sheet

```
mov    FSR,#$70    ;Select Bank 3
clr     $10         ;Clear register $10 on Bank 3
mov     FSR,#$D0    ;Select Bank 6
clr     $10         ;Clear register $10 on Bank 6
```

Indirect Addressing

To access any register via indirect addressing, simply move the 8-bit address of the register you wish to access into the FSR and use IND as the operand. This example will clear every RAM register on every bank using indirect addressing.

```
Temp    EQU    $08    ;Assign Temp as the alias
                        ;for register 8
INIT     mov     FSR,#Temp ;FSR = addr of Temp
LOOP     clr     IND      ;Clear register
          inc     FSR      ;Point to next register
          cjne    FSR,#0,LOOP ;Repeat until all registers
                        ;have been cleared
```

The Bank Instruction

Often it is desirable to set the bank select bits of the FSR with one instruction cycle (the MOV FSR, #literal commands above take two cycles). The SX instruction set offers such an instruction called Bank. The Bank instruction sets the upper bits of the FSR to point to the RAM bank required. Here's an example of how to use the Bank instruction:

```
Zero     EQU    $00
One       EQU    $30
Two       EQU    $50
Three     EQU    $70
Four      EQU    $90
Five      EQU    $B0
Six       EQU    $D0
Seven     EQU    $F0

bank      Three    ;Make FSR point to bank 3
clr       $10      ;Clear register $10 (in bank 3)
bank      Six      ;Make FSR point to bank 6
mov       $10,#1   ;Set register $10 (in bank 6) to 1
```

Appendix F: SX Data Sheet

Port Configuration Registers

Port A Registers

There are three registers used to configure the I/O pins of Port A. The TRIS_A register configures the data direction of the Port A pins as input or output. The LVL_A register configures the input pins as TTL or CMOS voltage level. The PLP_A register enables/disables pull up resistors on Port A input pins. To access these registers you must first write a particular value to the MODE register. Please refer to Table F.4 to find the values required in the MODE register to access the following Port A Registers. *Note: All the bits in the following registers are set to '1' on power up.*

TRIS_A – Data Direction Register

TRIS_A							
7	6	5	4	3	2	1	0
-	-	-	-	RA3	RA2	RA1	RA0

A bit set to '1' in this register sets the corresponding I/O port pin to input (high z) mode.

A bit set to '0' in this register sets the corresponding I/O port pin to output mode.

LVL_A - TTL/CMOS Select Register

LVL_A							
7	6	5	4	3	2	1	0
-	-	-	-	RA3	RA2	RA1	RA0

A bit set to '1' in this register sets the input level of the corresponding port pin to TTL.

A bit set to '0' in this register sets the input level of the corresponding port pin to CMOS.

SX Data Sheet

PLP_A – Pull-Up Resistor Enable Register

PLP_A							
7	6	5	4	3	2	1	0
-	-	-	-	RA3	RA2	RA1	RA0

A bit set to '1' in this register disables the weak pull-up resistor on the corresponding port pin.

A bit set to '0' in this register enables the weak pull-up resistor on the corresponding port pin.

Port B Registers

There are eight registers used to configure the I/O pins of Port B. The TRIS_B register configures the data direction of the Port B pins as input or output. The LVL_B register configures the input pins as TTL or CMOS voltage level. The PLP_B register enables/disables pull up resistors on Port B input pins. The ST_B register enables/disables the Schmitt-Trigger inputs on Port B input pins. The WKEN_B register enables/disables the multi-input wake up for interrupts on Port B input pins. The WKED_B register selects rising/falling edge detection on Port B input pins. The WKPND_B register contains the state of the MIWU pins. The CMP_B registers configures and provides the results from the comparator pins. To access these registers you must first write a particular value to the MODE register. Please refer to Table F.4 to find the values required in the MODE register to access the Port B Registers.

Note: All the bits in the following registers are set to '1' on power up.

Appendix F: SX Data Sheet

TRIS_B – Data Direction Register

TRIS_B							
7	6	5	4	3	2	1	0
RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0

A bit set to '1' in this register sets the corresponding I/O port pin to input (high z) mode.

A bit set to '0' in this register sets the corresponding I/O port pin to output mode.

LVL_B - TTL/CMOS Select Register

LVL_B							
7	6	5	4	3	2	1	0
RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0

A bit set to '1' in this register sets the input level of the corresponding port pin to TTL.

A bit set to '0' in this register sets the input level of the corresponding port pin to CMOS.

PLP_B – Pull-Up Resistor Enable Register

PLP_B							
7	6	5	4	3	2	1	0
RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0

A bit set to '1' in this register disables the weak pull-up resistor on the corresponding port pin.

A bit set to '0' in this register enables the weak pull-up resistor on the corresponding port pin.

SX Data Sheet

ST_B – Schmitt-Trigger Enable Register

ST_B							
7	6	5	4	3	2	1	0
RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0

A bit set to '1' in this register disables the Schmitt-Trigger input on the corresponding port pin.

A bit set to '0' in this register enables the Schmitt-Trigger input on the corresponding port pin.

WKEN_B – Wake Up Enable Register

WKEN_B							
7	6	5	4	3	2	1	0
RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0

A bit set to '1' in this register disables the multi-input wake up for the corresponding port pin.

A bit set to '0' in this register enables the multi-input wake up for the corresponding port pin.

WKED_B – Wake Up Edge Select Register

WKED_B							
7	6	5	4	3	2	1	0
RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0

A bit set to '1' in this register selects falling edge detection for the corresponding port pin.

A bit set to '0' in this register selects rising edge detection for the corresponding port pin.

Appendix F: SX Data Sheet

WKPND_B – MIWU Pending Register

WKPND_B							
7	6	5	4	3	2	1	0
RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0

A bit set to '1' in this register indicates a rising or falling edge was detected for the corresponding port pin.

A bit set to '0' in this register indicates no edge was detected on the corresponding port pin since that bit was last cleared.

CMP_B – Comparator Enable Register

CMP_B							
7	6	5	4	3	2	1	0
EN	OE	Rsvd	Rsvd	Rsvd	Rsvd	Rsvd	RES

- EN - Comparator Enable; 0 = enabled, 1 = disabled
- OE - Comparator Output Enable; 0 = enabled, 1 = disabled
- Rsvd - Reserved for future use
- RES - Comparator Result; (EN must = 0)

Port C Registers

There are four registers used to configure the I/O pins of Port C. The TRIS_C register configures the data direction of the Port C pins as input or output. The LVL_C register configures the input pins as TTL or CMOS voltage level. The PLP_C register enables/disables pull up resistors on Port C input pins. The ST_C register enables/disables the Schmitt-Trigger inputs on Port C input pins. To access these registers you must first write a particular value to the MODE register. Please refer to Table F.4 to find the values required in the MODE register to access the Port C Registers. *Note: All the bits in the following registers are set to '1' on power up.*

SX Data Sheet

TRIS_C – Data Direction Register

TRIS_C							
7	6	5	4	3	2	1	0
RC7	RC6	RC5	RC4	RC3	RC2	RC1	RC0

A bit set to '1' in this register sets the corresponding I/O port pin to input (high z) mode.

A bit set to '0' in this register sets the corresponding I/O port pin to output mode.

LVL_C - TTL/CMOS Select Register

LVL_C							
7	6	5	4	3	2	1	0
RC7	RC6	RC5	RC4	RC3	RC2	RC1	RC0

A bit set to '1' in this register sets the input level of the corresponding port pin to TTL.

A bit set to '0' in this register sets the input level of the corresponding port pin to CMOS.

PLP_C – Pull-Up Resistor Enable Register

PLP_C							
7	6	5	4	3	2	1	0
RC7	RC6	RC5	RC4	RC3	RC2	RC1	RC0

A bit set to '1' in this register disables the weak pull-up resistor on the corresponding port pin.

A bit set to '0' in this register enables the weak pull-up resistor on the corresponding port pin.

Appendix F: SX Data Sheet

ST_C – Schmitt-Trigger Enable Register

ST_C							
7	6	5	4	3	2	1	0
RC7	RC6	RC5	RC4	RC3	RC2	RC1	RC0

A bit set to '1' in this register disables the Schmitt-Trigger input on the corresponding port pin.

A bit set to '0' in this register enables the Schmitt-Trigger input on the corresponding port pin.

Control registers

There are three registers that configure the SX: MODE, OPTION, and Fuses. These registers allow the user to configure the SX in many ways. MODE and OPTION are run-time readable and writable while Fuses is written to only at program time.

Mode

The MODE register (simply called M in SX-Key mnemonics) is a run-time readable and writable register used to select the configuration registers for port operations.

MODE							
7	6	5	4	3	2	1	0
0	0	0	0	M3	M2	M1	M0

When a port configuration instruction is executed, such as MOV !RA, #1, the value of the MODE determines exactly which type of port configuration register will be written to. The right 4 bits in this register are set to '1' on power up. Below is an example detailing how the MODE register is used.

```
mov  M,$0F      ;Set up MODE for Direction config.
mov  !RA,$03     ;RA3:RA2 = Outputs, RA1:RA0 = Inputs
mov  M,$0E      ;Set up MODE for Pull-Up config.
mov  !RA,$01     ;RA3:RA1 = Normal,RA0 = Pull-up enabled
mov  M,$0D      ;Set up MODE for TTL/CMOS config.
mov  !RA,$02     ;RA3,RA2,RA0 = TTL, RA1 = CMOS
```

SX Data Sheet

MODE	Port A	Port B	Port C
\$0F	TRIS_A	TRIS_B	TRIS_C
\$0E	PLP_A	PLP_B	PLP_C
\$0D	LVL_A	LVL_B	LVL_C
\$0C		ST_B	ST_C
\$0B		WKEN_B	
\$0A		WKED_B	
\$09		Swap W with WKPEN_B	
\$08		Swap W with COMP_B	
\$07 - \$00			

Table F.4: Mode value definitions.

The table above defines the allowed mode values and their functions. The gray areas are undefined at this time.

Appendix F: SX Data Sheet

Option

The OPTION register is a run-time readable and writable register used to configure the RTCC and the Watchdog Timer. The size of this register is affected by the OPTION Extend bit as noted in Table F.6.

OPTION							
7	6	5	4	3	2	1	0
RTW	RTI	RTS	RTE	PSA	PS2	PS1	PS0

When OPTION Extend = 0, bits 7 and 6 are implemented.

When OPTION Extend = 1, bits 7 and 6 read as '1's.

- RTW - If = 0, register \$01 is W
If = 1, register \$01 is RTCC
- RTI - If = 0, RTCC roll-over interrupt is enabled
If = 1, RTCC roll-over interrupt is disabled
- RTS - If = 0, RTCC increments on internal instruction cycle
If = 1, RTCC increments on transition of RTCC pin
- RTE - If = 0, RTCC increments on low-to-high transition
If = 1, RTCC increments on high-to-low transition
- PSA - If = 0, prescaler is assigned to RTCC, divide rate determined by PS0-PS2 bits
If = 1, prescaler is assigned to WDT, and divide rate on RTCC is 1:1

Table F.5: Prescaler division ratios.

PS2, PS1, PS0	RTCC Divide Rate	Watchdog Timer Divide Rate
000	1:2	1:1
001	1:4	1:2
010	1:8	1:4
011	1:16	1:8
100	1:32	1:16
101	1:64	1:32
110	1:128	1:64
111	1:256	1:128

Fuses

The Fuses register is accessible only at program time. The SX-Key Development System Software provides a convenient interface that is

SX Data Sheet

easy to use to customize the SX. You may use the predefined device directives in your source code to specify the bits in the Fuses register. The table below depicts the bits of the Fuses register and describes their functions.

Parameter	Description	Values permitted
“Pins”	Number of pins on device	18, 28
“E ² Flash”	Number of 512-word E2Flash pages enabled	1, 2, 4, 8
“RAM Banks”	Number of RAM Banks enabled	1, 2, 4, 8
“Fuses”	Turbo Mode Stack Extend OPTION Extend ADD / SUB with C Input Syncing Watchdog Timer Code Protect	On / Off On / Off (8 / 2 Levels) On / Off (RTW & RTI available) On / Off (Carry used / ignored) On / Off (Inputs filtered / raw) On / Off On / Off
“Oscillator”	Oscillator Type and speed	HS, XT, RC, LP, Internal (user must select speed)
“Reset”	Select Brown Out reset	On / Off

Table F.6: Fuses Description.

Interrupts

Description

Sometimes a particular task or event must have the immediate attention of the processor. An interrupt is a means to accomplish this. In theory, the processor stops whatever it was doing and immediately begins executing code located at a special location called the Interrupt Vector. Once the code located at the Interrupt Vector task has completed, the processor returns to where it was before the interruption occurred.

In reality, several things must occur in addition to the aforementioned to ensure proper operation of the interrupt and the rest of the program. For one thing, consider the likely possibility that the interrupt occurred when the W register held a number the main program was using for a calculation. More than likely, the interrupt service routine will use the W register for its purposes too. When the processor finishes the interrupt service routine and returns to what it was doing before, the W register will hold a different value than what it held before the

Appendix F: SX Data Sheet

interrupt occurred. This can lead to bizarre program execution. In addition to the W register, the Status and FSR registers must be ‘preserved’ across an interrupt.

Traditionally, these issues were dealt with by software within the interrupt service routine. The engineers at Scenix had the foresight to take the burden off the programmer and put it in the chip where it belongs.

The Specifics

When an interrupt occurs in an SX chip, the STATUS, FSR, and W registers are saved in special shadow locations, and additional interrupts ignored. The program counter is loaded with \$00, (The Interrupt Vector), and the top three bits of the STATUS register, (PA2:PA0) are cleared to \$0. When the interrupt service routine has completed and the ‘RETI’ instruction is executed, the Status, FSR, and W registers are restored and the interrupt is re-enabled. Since this occurs automatically, your interrupt service routine does not have to waste any valuable time copying several registers back and forth.

RTCC Interrupt

The SX chip offers one internal interrupt called the RTCC rollover. If enabled, when the RTCC increments from \$FF to \$00, an interrupt will be generated. The latency, or response delay, will be exactly three instruction cycles in Turbo Mode, and exactly eight cycles in Compatible Mode. When the interrupt is complete, there will be a three-cycle delay (Turbo mode) or an eight-cycle delay (Compatible mode) before main code begins executing. This is due to the pipeline. Whenever an instruction is executed and, because of the instruction, the program counter is changed, the pipeline must be flushed and refilled. See RTCC Rollover Interrupts in Chapter 7 for more information.

RB0-RB7 Interrupt

The SX offers eight sources of external interrupts; a change of state on any of RB0 – RB7 can generate an interrupt. These can be individually configured via the WKEN_B and WKED_B configuration registers. The latency, or response delay, will be five cycles in Turbo mode and ten cycles in Compatible mode. See Wake-Up (Interrupt) on Edge Detect in Chapter 7 for more information.

SX Data Sheet

PERIPHERALS

Oscillator Driver

The SX chip offers a configurable oscillator driver that supports five types of oscillators:

LP:	Low Power Oscillator
XT:	Crystal or Resonator
HS:	High Speed Crystal or Resonator
RC:	Resistor/Capacitor
IRC:	Internal Resistor/Capacitor

XT, LP, and HS Mode

In XT, LP, and HS modes, a crystal or ceramic resonator can be connected to the SX chip as in Figures F.14 and F.15. The SX oscillator driver design requires the use of a parallel cut crystal. Use of a series cut crystal may give a frequency out of the crystal manufacturer's specifications. The values of the components can be determined from the Component Selection Tables below. Please note that some ceramic resonators have internal capacitors so that no external capacitors are required.

An external clock source can also be used to drive the OSC1 pin (leaving the OSC2 pin disconnected) as in Figure F.16.

Appendix F: SX Data Sheet

Figure F.14: SX with External Crystal.

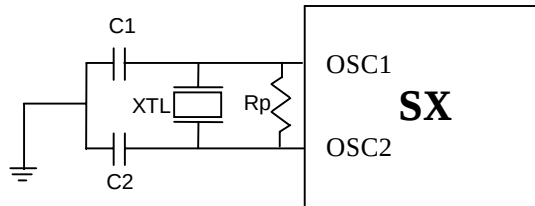
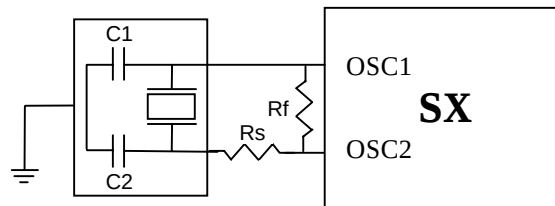


Table F.7: Component Selection for Crystals.

Type	Frequency	C1	C2	Rp
XT	4 MHz	20 pF	47 pF	1 M Ω
HS	8 MHz	20 pF	47 pF	1 M Ω
HS	12 MHz	20 pF	47 pF	1 M Ω
HS	16 MHz	15 pF	30 pF	1 M Ω
HS	20 MHz	15 pF	30 pF	1 M Ω
HS	25 MHz	5 pF	20 pF	10 k Ω
HS	30 MHz	5 pF	20 pF	4.7 k Ω
HS	36 MHz	5 pF	15 pF	3.3 k Ω
HS	40 MHz	5 pF	15 pF	3.3 k Ω
HS	50 MHz	5 pF	10 pF	3.3 k Ω

Figure F.15: SX with External Ceramic Resonator.



SX Data Sheet

Type	Frequency	C1	C2	Rp	Rs
XT	455 kHz	220 pF	220pF	1 M Ω	6.8 k Ω
XT	1 MHz	100 pF	100pF	1 M Ω	6.8 k Ω
XT	2 MHz	100 pF	100pF	100 k Ω	680 Ω
HS	4 MHz	100 pF	100pF	100 k Ω	0
HS	4 MHz	47 pF*	47 pF*	100 k Ω	470 Ω
HS	8 MHz	30 pF	30 pF	1 M Ω	0
HS	8 MHz	47 pF*	47 pF*	1 M Ω	470 Ω
HS	12 MHz	30 pF	30 pF	1 M Ω	0
HS	12 MHz	22 pF*	22 pF*	1 M Ω	0
HS	16 MHz	15 pF	15 pF	1 M Ω	0
HS	16 MHz	15 pF*	15 pF*	1 M Ω	0
HS	20 MHz	10 pF	10 pF	1 M Ω	0
HS	33 MHz	10 pF	10 pF	33 k Ω	0
HS	50 MHz	5 pF*	5 pF*	33 k Ω	0

* capacitors built in to resonator.

Table F.8: Component Selection for Ceramic Resonators.

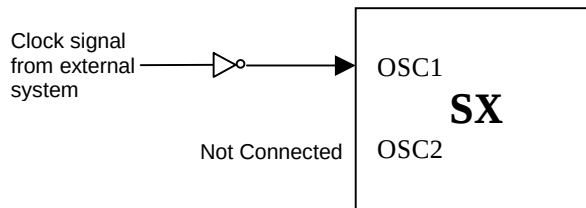


Figure F.16: SX with External system clock.

External RC Mode

For timing insensitive applications, the RC device option offers additional cost savings. The RC oscillator frequency is a function of the supply voltage, the resistor (Rext) and capacitor (Cext) values, and the operating temperature. In addition to this, the oscillator frequency will vary from unit to unit due to normal process variation. Furthermore, the difference in lead frame capacitance between package types will also affect the oscillation frequency, especially for low Cext values. Variation due to tolerance of external R and C components must also be considered.

Appendix F: SX Data Sheet

Figure F.17: SX with External RC clock.

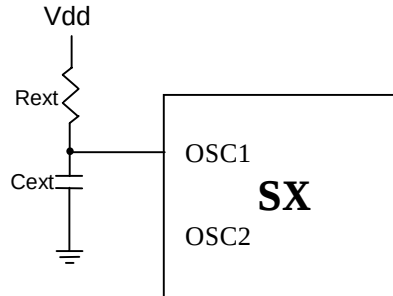


Figure F.17 shows the RC connections to the SX. For R_{ext} values below $2.2k\Omega$, the oscillator operation may become unstable, or stop completely. For very high R_{ext} values (e.g. $1\text{ M}\Omega$) the oscillator becomes sensitive to noise, humidity and leakage. The recommended R_{ext} value is $3k\Omega$ to $100k\Omega$.

Although the oscillator will operate with no external capacitor ($C_{ext} = 0\text{ pF}$), using values above 20 pF is recommended for noise and stability reasons. With little external capacitance, the oscillation frequency can vary dramatically due to changes in external capacitance, such as PCB trace capacitance or package lead frame capacitance.

Internal RC Mode

The SX offers an internal 4 MHz RC oscillator for timing insensitive operations. Using the internal oscillator reduces external component count and system cost. The internal oscillator is configured via fuses in the Fuses byte. The SX-Key Development software's Device screen allows you to select the internal RC oscillator speed division factor (1, 2, 4, 8, 16, 32, 64 or 128). A Device directive is also offered, allowing source code to have the proper fuse configuration embedded within it.

SX Data Sheet

@ sign, 74-76
; (comments), 35-36
= (assignment), 37,41
= (equal), 42
< (less than), 42
> (greater than), 42
<> (not equal), 42
=< (equal or less than), 42
=> (equal or greater than), 42
|| (absolute value), 53
- (negative), 53
~ (not), 53
+ (addition), 53
- (subtraction), 53
* (multiplication), 53
/ (division), 53
// (modulus), 53
& (logical AND), 53
| (logical OR), 53
^ (logical XOR), 53
<< (shift left), 53
>> (shift right), 53
<> (reverse bits), 53
% (iteration symbol), 43-44
\ (argument symbol), 45
b (bit address), 84
f (file register), 84
k (constant), 84
!port, 84

- A -

ABSOLUTE VALUE (||), 53
ADD, 85
ADDB, 85
ADDITION (+), 53
ADJUSTING INT. CYCLES, 69-72
AND, 85

AND (&) (logical), 53
ARGUMENT SYMBOL (\), 45
ASCII DATA TYPE, 54
ASSEMBLER DIRECTIVES, 35-50
= (assignment), 37,41
BREAK, 37,47
CASE, 37,46-47
DEVICE, 37,38
DS, 37,41
DW, 37,41
ELSE, 37,41-43
END, 37
ENDIF, 37,41-43
ENDM, 37,44-45
ENDR, 37,43-44
ERROR, 37,50
EXITM, 37,44-45
EXPAND, 37,46
EQU, 37,41
FREQ, 37,40
ID, 37,40
IF..ELSE..ENDIF, 37,41-42
IFDEF..ELSE..ENDIF, 37,41-42
IFNDEF..ELSE..ENDIF, 37,42-43
MACRO, 37,44-45
NOCASE, 37,46-47
NOEXPAND, 37,46
ORG, 37,40
REPT, 37, 43-44
RESET, 37,40
WATCH, 37,47-49
ASSEMBLY, 35, 41-43
Conditional, 41-43
Program, typical, 35
ASSEMBLY CODE BOX, 26,27,30
ASSIGNMENT (=), 37,41

Index

- B -

BACKSPACE KEY, 29
BANK, 86,136
BANKS1 (param.), 38
BANKS2 (param.), 38
BANKS4 (param.), 38
BINARY DATA TYPE, 54
BINARY OPERATORS (table), 53
BREAK (directive), 37,47
BREAKPOINTS, 30,47
 setting, 30,47
BROWNOUT (param.), 38

- C -

C (carry), (table) 84, 133
CALIBRATING IRC, 33
CALL, 74-76,86,129
CALLING
 Across pages, 74-75,129-130
CARRYX (param.), 38,82,84
CASE (directive), 37,46-47
CASE SENSITIVITY 46-47
CJA, 86
CJAE, 87
CJB, 87
CJBE, 88
CJE, 88
CJNE, 87
CLC, 89
CLR, 89
CLRB, 89,132
CLZ, 90
CMOS/TTL LOGIC
 (table), 58
 Configuration,55,58-59
CMP_B, 64

CODE BUTTON, 28
CODE PAGES, 73-76
CODE PROGRAM (param.), 33
COMMENTS (;), 35-36
COMPARATOR CONFIG., 55, 64-65
COMPARATOR REGISTER, 141
COMPARISON OPERATORS (table), 42
CONDITIONAL 41-43
 Assembly, 41-43
 Statements, 42-43
CONFIGURE WINDOW, 32-33
 Code Program param., 33
 Device Erase param., 33
 Fuse Program param., 33
 IRC Calibration, 33
CONTROL REGISTERS, 143-146
CRYSTAL COMPONENTS (table), 149
CSA, 90
CSAE, 90
CSB, 91
CSBE, 91
CSE, 92
CSNE, 92
CURRENT INSTRUCTION, 30
 Setting, 30
CURSOR KEYS, 29

- D -

DATA DIRECTION REG.,137,139,142
 TRIS_A, 137
 TRIS_B, 139
 TRIS_C, 142
DATA TABLES, 71-73
 8-bit, 71-72
 12-bit, 72-73
DATA TYPES, 54
 Size, 54

- DC** (digit carry), (table) 84,133
- DEBUG BUTTONS** (figure), 26
- DEBUG WINDOW**, 26-27
- DEBUG WINDOWS**, 25-30
 - (figure), 26
 - Editing keys (table), 29
- DEBUGGING**, 25-30
 - requirements, 30
- DEC**, 92
- DECIMAL DATA TYPE**, 54
- DECSZ**, 93
- DEFINE SPACE** (directive), 37,41
- DEFINE WORD** (directive), 37,41
- DEMO BOARD, SX-KEY**,9-11,115-118
 - Configuring Hyperterminal, 11
 - Configuring Virtual Periph.,117-118
 - Connections, 10
 - Connecting SX-Key, 10
 - Features, 115-116
 - Plugging SX into, 9-10
 - Schematic, 118
 - Sxdemo.src, 10
 - Virtual Peripheral Demo, 11
- DETECTING EDGES**, 60-64
- DEVICE** (directive) 37-39
 - Compatible (table), 39
 - HS_OSC, 39
 - LP_OSC, 39
 - RC_OSC, 39
 - XT_OSC, 39
 - PIC16C54, 39
 - PIC16C54A, 39
 - PIC16C55, 39
 - PIC16C56, 39
 - PIC16C57, 39
 - PIC16C58, 39
 - PIC16C58A, 39
 - PROTECT_OFF, 39
 - PROTECT_ON, 39
 - WDT_OFF, 39
 - WDT_ON, 39
 - Settings (table), 38
- DEVICE ERASE** (param.), 33
- DEVICE WINDOW**, (fig) 31-32
 - Load Hex button, 32
 - Options, 31
 - Oscillator, 31
 - Frequency, 40
 - Internal, 33-34
 - Program button, 32
 - Read button, 32
 - Save Hex button, 32
 - Verify button, 32
- DIFFERENCES FROM PICS**, 17-18
- DIRECT/INDIRECT ADDR.**, 127,128
- DIRECTION CONFIG., PORT**, 55-57
- DIRECTIVES**, 35-50
 - (table), 37
- DISABLING INTERRUPTS, AUTO**, 66
- DISABLING MACRO EXPAND**, 46
- DISCUSSION LIST**, ii
- DIVISION (/)**, 53
- DJNZ**, 93
- DS** (directive), 37,41
- DW** (directive), 37,41, 72-73
-
- E -**
-
- E²Flash**, 31,32
- EDGE DETECTION**
 - Configuration, 55,60-62
 - Interrupts, 55,62-64,148
- EDGE SELECTION REG.**, 60-64
- EDIT MENU**, 23
- EDITING KEYS, DEBUG** (table), 29
- EDITOR**, 21-34
 - (figure), 21

Index

Copy, 22
Cut, 22
Editing shortcut keys (table), 22
Menus, 22-24
 Edit, 23
 File, 22
 Help, 24
 Run, 23
Page down, 22
Page up, 22
Paste, 22
Tab, 22
Text bigger, 22
Text sizing, 22
Text smaller, 22
Windows, 24-34
ELSE (directive), 37,41-43
END (directive), 37
ENDIF (directive), 37,41-43
ENDM (directive), 37,44-45
ENDR (directive), 37,43-44
ENTER KEY, 29
EMAIL, SUPPORT, ii
EQU (directive), 37,41
EQUAL (=) (operator), 42
EQUAL OR LESS THAN (= <) (operator), 42
EQUAL OR GREATER THAN (=>) (operator), 42
ERRATA SHEET, 5
ERROR (directive), 37,50
ERROR MESSAGES, 109-114
ESC KEY, 29
EXITM (directive), 37,44-45
EXPAND (directive), 37,46
EXPANDING MACROS, 46
EXPRESSIONS, 52-53
EXTERNAL RC MODE, 150-151

- F -

FILE MENU, 22
FILE SELECT REGISTER, 134-136
FIND TEXT, 25
FIND/REPLACE TEXT, 25
FORMAT SETTING, WATCH
 (table), 48
 Watch (directive), 47-49
FR, 84
FREQ (directive), 30,37,40
FSR, 84,126, 134-136, (table) 135
FSR INDICATOR, 26
FSTR (formatter), 47,49
FTP SITE, ii
FUSE PROGRAM (param.), 33
FUSES, 146

- G -

GLOBAL LABELS, 51-52
GREATER THAN (>) (operator), 42

- H -

HELP MENU, 24
HEX FILES, 32
 Loading, 32
 Saving, 32
HEXADECIMAL DATA TYPE, 54
HOP BUTTON, 27,28
HS MODE, 148
HS_OSC (param.), 39

- I -

I/O PINS, 31, 55-65
ID (param.), 32
ID (directive), 37,40
IND, 126,127
INDIRECT ADDRESSING, 136
INPUT/OUTPUT CONFIG., 55-57
INTERNAL RC MODE, 151
IF..ELSE..ENDIF (directive), 37,41-42
IFDEF..ELSE..ENDIF (directive), 37,41-42
IFNDEF..ELSE..ENDIF (directive), 37,42-43
IJNZ, 93
INC, 93
INCSZ, 94
INSTRUCTION
 Multi-word (table), 81
 Pipeline, 125-126
 Quick ref. (table), 82
 Single-word (table), 80
 Summary (table), 79
INT FLAG, 29
INTERRUPT
 Adjusting cycles, 69-71
 Debugging, 67
 Latency delays, 66
 Multiple, 67
 Queueing, 67
 Routine size, 67
 Vector, 66
INTERRUPTS, 62-64, 65-71, 146-148
IRC, 33-34
 Calibration, 33-34
IRC CALIBRATION, 34
IRCDIV1 (param.), 38
IRCDIV2 (param.), 38

IRCDIV4 (param.), 38
IRCDIV8 (param.), 38
IRCDIV16 (param.), 38
IRCDIV32 (param.), 38
IRCDIV64 (param.), 38
IRCDIV128 (param.), 38
IREAD, 70,72-73,94
ITERATION SYMBOL (%), 43-44

- J -

JOG BUTTON, 27,28
JUMP, 73-74,128
JUMPING ACROSS PAGES, 73-74,128

- L -

LABELS, 35,51-52
 Global, 51-52
 Local, 51-52
 Macro, 51-52
LESS THAN (<) (operator), 42
LIST GENERATION, 22
LIST TOGGLE, 22
LOAD HEX BUTTON, 32
LOADING OBJECT FILES, 32
LOCAL LABELS, 51-52
LOGIC LEVEL CONFIG., 55,58-59
LP MODE, 148
LP_OSC (param.), 39
JB, 94
JMP, 95
JNB, 95
JNC, 95
JZ, 96
LCALL, 96
LJMP, 97

Index

LSET, 97

LVL_A, 137

LVL_B, 139

LVL_C, 142

- M -

M, 84

MACRO, 37, 44-46, 51-52

(directive), 37,44-45

Expanding, 46

Labels, 51-52

MNEMONICS, 35

MODE, 97

MODE REGISTER, 55-56,143

MODE VALUES (table), 56,144

MODIFYING REGISTERS, 29

MODULUS (/), 53

MOV, 98

MOVB, 99

MOVSZ, 99

MULTI-WORD INST. (table), 81

MULTIPLICATION (*), 53

- N -

NEGATIVE (-), 53

NOCASE (directive), 37,46-47

NOEXPAND (directive), 37,46

NOP, 99

NOT, 100

NOT (~) (operator), 53

NOT EQUAL (<>) (operator), 42

- O -

OBJECT FILES, 32

Loading, 32

Saving, 32

OE (bit), 65

OP.BIT, 84

OPERATORS, 42,53

Binary, 53

Conditional, 42

Unary, 53

OPT, 84

OPTION REGISTER, 145

OPTIONS, DEVICE WINDOW, 32

OPTIONX (param.), 38, 68

OR, 100

OR, (|) (logical), 53

ORG (directive), 37,40

OSCHS (param.), 38

OSCILLATOR DRIVER, 148

OSCLP (param.), 38

OSCRC (param.), 38

OSCXT (param.), 38

OSCILLATOR, DEVICE WINDOW, 32

Frequency, 40

Internal, 33-34

- P -

PACKING LIST, 5-6

Skeleton Key, 5

Master Key 18, 6

Master Key 28, 6

PAGE, 74,100,132

PAGES1 (param.), 38

PAGES2 (param.), 38

PAGES4 (param.), 38

PARALLAX INSTR. SET, 79-106

- PC**, 84,126,128
- PD** (power down), 84,133
- PENDING REGISTERS**, 60-64,67,141
- PERIPHERALS**, 148-151
- PIC16C54** (param.), 39
- PIC16C54A** (param.), 39
- PIC16C55** (param.), 39
- PIC16C56** (param.), 39
- PIC16C57** (param.), 39
- PIC16C58** (param.), 39
- PIC16C58A** (param.), 39
- PICS**, 17-18
 - Emulation, 17-18
 - Fuses, 18
 - Similarities with, 17
 - Differences, 17-18
 - I/O pins, 18
 - Instruction execution, 17,18
 - Interrupts, 18
 - Oscillator, 18
 - Pinout, 17
 - Opcodes, 17
 - Program space, 17
 - RAM registers, 17
 - Stack, 18
- PINS**, 31,55-65
 - I/O, 31,55-65
- PINS18** (param.), 38
- PINS28** (param.), 38
- PLP_A**, 138
- PLP_B**, 139
- PLP_C**, 142
- POLL BUTTON**, 28,29
- POLLING**, 27-28
- POP**, 131
- PORT A REGISTERS**, 137-138
- PORT B REGISTERS**, 138-141
- PORT C REGISTERS**, 141-143
- PORT CONFIG. REGISTERS**, 137-143
- PORT CONFIGURATION**, 55-56
- PRESCALER DIVISION RATIO**, 145
- PRINT WINDOW**, 24
- PRINTING**, 24
- PROGRAM BUTTON**, 32
- PROGRAM COUNTER** (PC), 28, 30
 - Setting, 30
- PROTECT** (param.), 38
- PROTECT_OFF** (param.), 39
- PROTECT_ON** (param.), 39
- PULL-UP RESISTER CONFIG.**,55,57-58
- PULL-UP RESISTOR REG.**,138,139,142
 - PLP_A, 138
 - PLP_B, 139
 - PLP_C, 142
- PUSH**, 130-131
-
- Q -**
-
- QUICK PROTO BOARD**, 12-13, 119-121
 - Connecting SX-Key, 13
 - Connections (figure), 12
 - Features, 119-120
 - Plugging SX into, 13
 - Schematic, 121
- QUICK START**, 9-13
 - SX-Key Demo Board, (fig) 9,9-12
 - SX-Key Quick Proto Board, (fig) 12, 12-13
-
- R -**
-
- RA**, 126
- RAM BANKS**, 31
- RAM REGISTER MAP**, 124, (figure),135
- RB**, 126
- RC**, 126
- RC_OSC** (param.), 39
- READ BUTTON**, 32

Index

READ-MODIFY-WRITE, 126
READING, 32
REGISTERS, 27-29
 Binary representation of, 27
 Modifying, 29
 Undoing modifications, 29
REGISTERS BUTTON, 28
REGISTERS WINDOW
 (figure), 26, 26-27
REPEAT (REPT) (directive), 37,43-44
REPLACE TEXT, 25
REPT (directive), 37, 43-44
REQUIREMENTS, 15,25-30
 Debugging, 25-30
 System, 15
RES (bit), 64-65
RESERVED WORDS (table), 107
RESET (directive), 37,40
RESET BUTTON, 27,28
RESONATOR COMPONENTS
 (table),150
RET, 100,132
RETI, 68-70,101,132
RETIW, 68-70,101,132
RETP, 101,132
RETW, 71-72,73
RETURNS, 132
RETURNING FROM SUBROUTINE, 130
RETW, 101,132
REVERSE BITS (<>), 53
RL, 102,133-134
ROTATE LEFT, 102, 133-134
ROTATE RIGHT, 102,133-134
RR, 102,133-134
RTCC, 126,127,132
RTCC INTERRUPTS, 69-71,147
RTE, 68
RTI, 68
RTS, 68

RUN BUTTON, 27,28
RUN MENU, 23

- S -

SAVE HEX BUTTON, 32
SAVING OBJECT FILES, 32
SB, 102
SC, 102
SCHMITT-TRIGGER
 (table), 59
 Configuration.,55,59-60
 Registers., 140, 143
 ST_B, 140
 ST_C, 143
SBIN (formatter), 48
SDEC (formatter), 48
SHEX (formatter), 48
SETB, 103,132
SETTING BREAKPOINTS, 30,47
SETTING CURRENT INSTRUCTION, 30
SHIFT LEFT (<<), 53
SHIFT RIGHT (>>), 53
SIMILARITIES WITH PICS, 17
SINGLE-WORD INSTR. (table), 80
SKIP, 103
SKIP FLAG, 29
SKIP INSTRUCTIONS, 84
SLEEP, 62,64,65,71,103
SNB, 103
SNC, 104
SNZ, 104
SOURCE CODE WINDOW, 26
SPACE KEY, 29
SPECIAL FUNCTION REG., 126-128
ST_B, 59-60
ST_C, 59-60
STACK, 130

- STACK OVERFLOW**, 131-132
- STACK UNDERFLOW**, 132
- STACKX** (param.), 38
- STATUS**, 126
- STATUS REGISTER**, 132-134
- STC**, 104
- STEP BUTTON**, 27,28
- STOP BUTTON**, 27,28
- STZ**, 104
- SUB**, 105
- SUBB**, 105
- SUBTRACTION (-)**, 53
- SUPER PARALLAX ASSEMBLER**
 - @ Sign, 19-20
 - Device settings, 20
 - Differences, 19-20
 - Directives and equates, 19
 - Environment, 19
 - Language syntax, 19
 - Mnemonics, 19
 - New mnemonics, 19
 - Similarities with, 19
- SUPPORT**, ii, 5
 - Discussion List, ii
 - Email, ii
 - FTP Site, ii
 - Web Site, ii, 5
- SWAP**, 105
- SX ARCHITECTURE**, 124-125
 - Instruction pipeline, 125-126
- SX CHIP FEATURES**, 77-78
- SX DATA SHEET**, 123-151
 - C (carry), 133
 - Comparator register, 141
 - Control registers, 143-146
 - Crystal components (table), 149
 - DC (digit carry), 133
 - Data direction registers, 137,139,142
 - TRIS_A, 137
 - TRIS_B, 139
 - TRIC_C, 142
- TTL/CMOS Selection reg., 137,139,142
 - LVL_A, 137
 - LVL_B, 139
 - LVL_C, 142
- Direct/Indirect Addressing, 127,128
- External RC mode, 150-151
- File Select Register, 134-136
- FSR, 126, 134-136, (table) 135
- Fuses, 146
- HS mode, 148
- IND, 126,127
- Indirect addressing, 136
- Internal RC mode, 151
- LP mode, 148
- Mode register, 143
- Option register, 145
- Oscillator driver, 148
- PC, 126,128
- PD (power down), 133
- Pending registers., 141
- Peripherals, 148-151
- POP, 131
- Port A registers, 137-138
- Port B registers, 138-141
- Port C registers, 141-143
- Port config. registers, 137-143
- Prescaler division ratio, 145
- Pull-Up resistor reg., 138,139,142
 - PLP_A, 138
 - PLP_B, 139
 - PLP_C, 142
- Push, 130-131
- RA, 126
- RB, 126
- RC, 126
- Read-modify-write, 126
- Resonator components (table), 150

Index

- Returns, 132
- Returning from a subroutine, 130
- Rotate left, 102, 133-134
- Rotate right, 102, 133-134
- RTCC, 126, 127, 132
- Schmitt-Trigger reg., 140, 143
 - ST_B, 140
 - ST_C, 143
- Special function registers, 126-128
- Stack, 130
- Stack overflow, 131-132
- Stack underflow, 132
- Status, 126
- Status register, 132-134
- TO (time out), 84, 133
- Wake-Up Enable reg., 140
- Wake-Up Edge reg., 140
- WREG, 127
- XT mode, 148
- Z (zero bit), 133
- SX-KEY ERROR MESSAGES**, 109-114
- SX-KEY HARDWARE CONNECTION**
 - Clock source, 7
 - Debugging, 7
 - Four-pin connector (figure), 7
 - Power, 7
 - Programming, 7, 32
- SX-KEY INTERFACE**, 21-34
- SYMBOLS**, 50-51
- SYNC** (param.), 38
- SYSTEM REQUIREMENTS**, 15
- SZ**, 106
- T -**
- TAB**, 29
- TEST**, 106
- TO** (time out), 133
- TRIS_A**, 137
- TRIS_B**, 139
- TRIS_C**, 142
- TRIS REGISTERS**, 56
- TTL/CMOS SELECTION REG.**, 137, 139, 142
 - LVL_A, 137
 - LVL_B, 139
 - LVL_C, 142
- TTL LOGIC LEVEL** (table), 58
- TURBO** (param.), 38
- U -**
- UBIN** (formatter), 48
- UDEC** (formatter), 48
- UHEX** (formatter), 48
- UNARY OPERATORS** (table), 53
- V -**
- VERIFY BUTTON**, 32
- VERIFYING**, 32
- VERSIONS**, ii
- VIRTUAL PERIPHERALS**, 11
- W -**
- W**, 84
- WAKE-UP ENABLE REG.**, 140
- WAKE-UP EDGE REG.**, 140
- WAKE-UP INTERRUPTS**, 55, 62-64
- WALK BUTTON**, 27, 28
 - Communicating with, 11
- WARRANTY**, ii
- WATCH** (directive), 37, 47-49
- WATCH BUTTON**, 28
- WATCH WINDOW**, 26, 29, 48
 - Modifying values, 29
- WATCHDOG** (param.), 38

WATCHING REGISTERS, 28, 47-49

WEBSITE, ii, 5

WKED_B, 60,62

WKPEN_B, 60,62

WREG, 127

- X -

XOR, 106

XOR (^) (logical), 53

XT MODE, 148

XT_OSC (param.), 39

- Z -

Z (zero bit), 84,133

ZSTR (formatter), 48,49

Index
